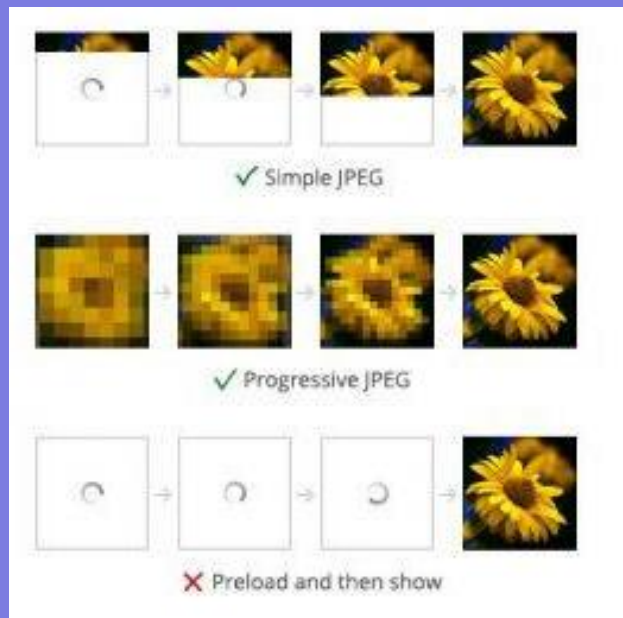


JPMC BOOTCAMP

SESSION 1

Dr. Ernesto Lee

WHY THIS IS DIFFERENT



ADEPT Method for Learning

Analogy	Tell me what it's like.
Diagram	Help me visualize it.
Example	Allow me to experience it.
Plain English	Describe it with everyday words.
Technical Definition	Discuss the formal details.

DATA

https://raw.githubusercontent.com/fenago/pythonml/main/data/WA_Fn-UseC_Telco-Customer-Churn.csv

WHAT YOU WILL LEARN

- Doing exploratory data analysis for identifying important features
- Encoding categorical variables to use them in machine learning models
- Using logistic regression for classification

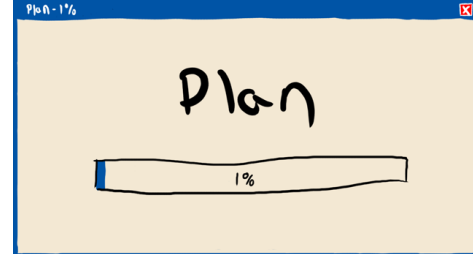


CHURN PROJECT

Are customers going to leave the company?



THE PLAN FOR THE PROJECT



1. First, we download the dataset and do some initial preparation: rename columns and change values inside columns to be consistent throughout the entire dataset.
2. Then we split the data into train, validation, and test so we can validate our models.
3. As part of the initial data analysis, we look at feature importance to identify which features are important in our data.
4. We transform categorical variables into numeric so we can use them in the model.
5. Finally, we train a logistic regression model.

THE DATASET



- **Services of the customers** – phone; multiple lines; internet; tech support and extra services such as online security, backup, device protection, and TV streaming
- **Account information** – how long they have been clients, type of contract, type of payment method
- **Charges** – how much the client was charged in the past month and in total
- **Demographic information** – gender, age, and whether they have dependents or a partner
- **Churn** – yes/no, whether the customer left the company within the past month

<https://www.kaggle.com/blastchar/telco-customer-churn>

INITIAL DATA PREPARATION

DO YOUR IMPORTS

```
import pandas as pd
```

```
import numpy as np
```

```
import seaborn as sns
```

```
from matplotlib import pyplot as plt
```

```
%matplotlib inline
```



READ THE DATASET

```
df = pd.read_csv('WA_Fn-UseC_-Telco-Customer-Churn.csv')
```

```
len(df)
```

```
df.head()
```

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	...
0	7590-VHVEG	Female	0	Yes	No	1	No	No phone service	DSL	No	...
1	5575-GNVDE	Male	0	No	No	34	Yes	No	DSL	Yes	...
2	3668-QPYBK	Male	0	No	No	2	Yes	No	DSL	Yes	...
3	7795-CFOCW	Male	0	No	No	45	No	No phone service	DSL	Yes	...
4	9237-HQITU	Female	0	No	No	2	Yes	No	Fiber optic	No	...

TRANSPOSE THE DATASET TO MAKE IT WIDE (NOT LONG)

`df.head().T`

	0	1	2
customerID	7590-VHVEG	5575-GNVDE	3668-QPYBK
gender	Female	Male	Male
SeniorCitizen	0	0	0
Partner	Yes	No	No
Dependents	No	No	No
tenure	1	34	2
PhoneService	No	Yes	Yes
MultipleLines	No phone service	No	No
InternetService	DSL	DSL	DSL
OnlineSecurity	No	Yes	Yes
OnlineBackup	Yes	No	Yes
DeviceProtection	No	Yes	No
TechSupport	No	No	No
StreamingTV	No	No	No
StreamingMovies	No	No	No
Contract	Month-to-month	One year	Month-to-month
PaperlessBilling	Yes	No	Yes
PaymentMethod	Electronic check	Mailed check	Mailed check
MonthlyCharges	29.85	56.95	53.85
TotalCharges	29.85	1889.5	108.15
Churn	No	No	Yes

DATA TYPES

```
df.dtypes
```

```
total_charges =  
pd.to_numeric(df.TotalCharges,  
errors='coerce')
```

df.dtypes

customerID	object
gender	object
SeniorCitizen	int64
Partner	object
Dependents	object
tenure	int64
PhoneService	object
MultipleLines	object
InternetService	object
OnlineSecurity	object
OnlineBackup	object
DeviceProtection	object
TechSupport	object
StreamingTV	object
StreamingMovies	object
Contract	object
PaperlessBilling	object
PaymentMethod	object
MonthlyCharges	float64
TotalCharges	object
Churn	object
dtype:	object

SeniorCitizen is integer

TotalCharges is not correctly
identified as a numeric type
(float or int)

CONSIDER EMPTIES...

```
df[total_charges.isnull()][['customerID', 'TotalCharges']]  
  
df.TotalCharges = pd.to_numeric(df.TotalCharges,  
errors='coerce')  
  
df.TotalCharges = df.TotalCharges.fillna(0)
```

COLUMN NAMES AND NAMING CONVENTIONS

```
df.columns = df.columns.str.lower().str.replace(' ', '_')  
string_columns = list(df.dtypes[df.dtypes == 'object'].index)  
for col in string_columns:  
    df[col] = df[col].str.lower().str.replace(' ', '_')
```

ENCODE CHURN TARGET VARIABLE

```
df.churn = (df.churn == 'yes').astype(int)
```

SPLIT THE DATA FOR TESTING AND TRAINING

```
from sklearn.model_selection import train_test_split  
  
df_train_full, df_test = train_test_split(df, test_size=0.2,  
random_state=1)
```

	customerid	gender	tenure
0	7590-vhveg	female	1
1	5575-gnvde	male	34
2	3668-qpybk	male	2
3	7795-cfocw	male	45
4	9237-hqitu	female	2
5	9305-cdskc	female	8
6	1452-kiovk	male	22
7	6713-okomc	female	10
8	7892-pookp	female	28
9	6388-tabgu	male	62



	customerid	gender	tenure
8	7892-pookp	female	28
2	3668-qpybk	male	2
5	9305-cdskc	female	8
6	1452-kiovk	male	22
3	7795-cfocw	male	45
1	5575-gnvde	male	34
0	7590-vhveg	female	1
7	6713-okomc	female	10
4	9237-hqitu	female	2
9	6388-tabgu	male	62



Train			
	customerid	gender	tenure
8	7892-pookp	female	28
2	3668-qpybk	male	2
5	9305-cdskc	female	8
6	1452-kiovk	male	22
3	7795-cfocw	male	45
1	5575-gnvde	male	34
0	7590-vhveg	female	1
7	6713-okomc	female	10

	customerid	gender	tenure
4	9237-hqitu	female	2
9	6388-tabgu	male	62

Shuffle

Split 80%/20%

Test

TRAIN, TEST, VALIDATE

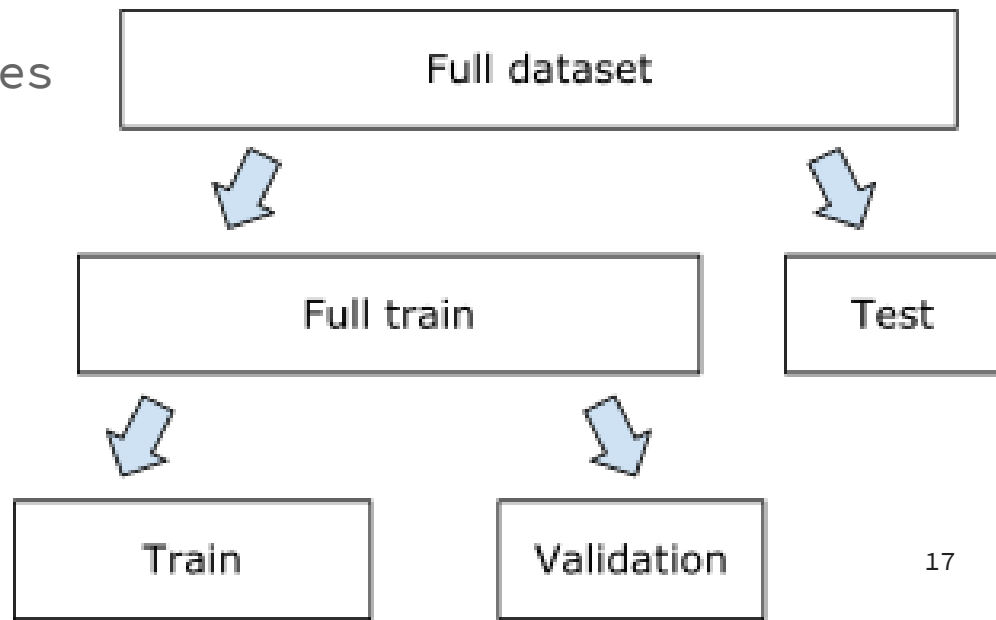
```
df_train, df_val = train_test_split(df_train_full,  
test_size=0.33, random_state=11)
```

```
y_train = df_train.churn.values
```

```
y_val = df_val.churn.values
```

```
del df_train['churn']
```

```
del df_val['churn']
```



EXPLORATORY DATA ANALYSIS

VALIDATE THERE ARE NO MISSING VALUES

```
df_train_full.isnull().sum()
```

```
df_train_full.isnull().sum()
```

customerid	0
gender	0
seniorcitizen	0
partner	0
dependents	0
tenure	0
phoneservice	0
multiplelines	0
internetservice	0
onlinesecurity	0
onlinebackup	0
deviceprotection	0
techsupport	0
streamingtv	0
streamingmovies	0
contract	0
paperlessbilling	0
paymentmethod	0
monthlycharges	0
totalcharges	0
churn	0
dtype: int64	

VALIDATE THE DISTRIBUTION OF THE TARGET VARIABLE

```
df_train_full.churn.value_counts()
```

What percentage of customers STOPPED using the services?

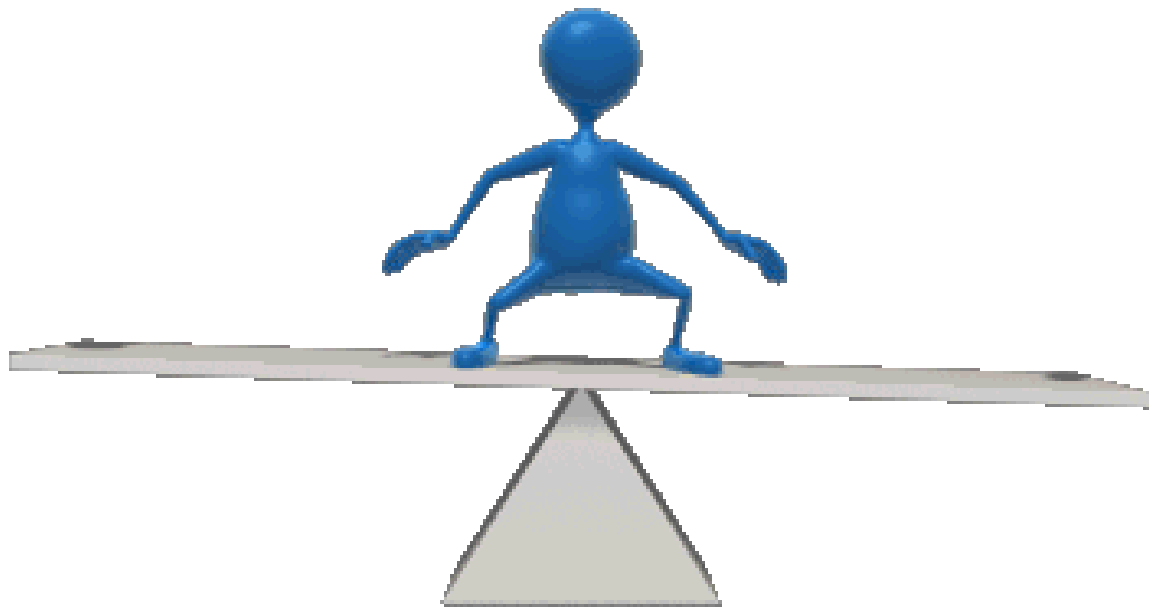
COMPUTE THE MEAN OF THE TARGET VARIABLE

```
global_mean = df_train_full.churn.mean()
```

```
global_mean = df_train_full.churn.mean()  
round(global_mean, 3)
```

```
0.27
```

IMBALANCED DATASET



CATEGORICAL & NUMERICAL COLUMNS REQUIRE DIFFERENT TREATMENTS

- **categorical**, which will contain the names of categorical variables,
- **numerical**, will have the names of numerical variables

```
categorical = ['gender', 'seniorcitizen', 'partner', 'dependents',  
              'phoneservice', 'multiplelines', 'internetservice',  
              'onlinesecurity', 'onlinebackup', 'deviceprotection',  
              'techsupport', 'streamingtv', 'streamingmovies',  
              'contract', 'paperlessbilling', 'paymentmethod']  
  
numerical = ['tenure', 'monthlycharges', 'totalcharges']
```

CATEGORICAL DATA

```
df_train_full[categorical].nunique()
```

```
df_train_full[categorical].nunique()
```

```
gender                2
seniorcitizen          2
partner               2
dependents            2
phoneservice          2
multiplelines         3
internetservice       3
onlinesecurity        3
onlinebackup          3
deviceprotection      3
techsupport           3
streamingtv           3
streamingmovies       3
contract              3
paperlessbilling      2
paymentmethod         4
dtype: int64
```


NUMERICAL DATA

Get the Descriptive statistics for each column (Univariate Analysis)

```
df_train_full[numerical].describe()
```

CORRELATIONS

```
df_train_full.corr()
```

FEATURE IMPORTANCE

FEATURE IMPORTANCE

- Knowing how other variables affect the target variable, churn, is the key to understanding the data and building a good model.
 - This process is called **feature importance** analysis
- We have two different kinds of features: categorical and numerical.
 - Each kind has different ways of measuring feature importance, so we will look at each separately.

FEATURE IMPORTANCE: CATEGORICAL VARIABLES

	customerid	gender	churn
0	7590-vhveg	female	0
1	5575-gnvde	male	0
2	3668-qpybk	male	1
3	7795-cfocw	male	0
4	9237-hqitu	female	1
5	9305-cdskc	female	1
6	1452-kiovk	male	0
7	6713-okomc	female	0
8	7892-pookp	female	1
9	6388-tabgu	male	0



gender == "female"

	customerid	gender	churn
0	7590-vhveg	female	0
4	9237-hqitu	female	1
5	9305-cdskc	female	1
7	6713-okomc	female	0
8	7892-pookp	female	1



	customerid	gender	churn
1	5575-gnvde	male	0
2	3668-qpybk	male	1
3	7795-cfocw	male	0
6	1452-kiovk	male	0
9	6388-tabgu	male	0

gender == "male"

STRATEGY FOR CATEGORICAL FEATURE IMPORTANCE

- So we can look at all the distinct values of a variable.
 - for each variable, there's a group of customers: all the customers who have this value.
 - For each such group, we can compute the churn rate, which will be the group churn rate.
 - When we have it, we can compare it with the global churn rate – churn rate calculated for all the observations at once.
- If the difference between the rates is small, the value is not important when predicting churn because this group of customers is not really different from the rest of the customers.
- On the other hand, if the difference is not small, something inside that group sets it apart from the rest.

An Analyst or machine learning algorithm should be able to pick this up and use it when making predictions.

FEATURE IMPORTANCE BASED ON GENDER

Let's check first for the gender variable. To compute the churn rate for all female customers, we first select only rows that correspond to `gender == 'female'` and then compute the churn rate for them:

```
female_mean = df_train_full[df_train_full.gender ==  
'female'].churn.mean()
```

```
male_mean = df_train_full[df_train_full.gender ==  
'male'].churn.mean()
```

FEATURE IMPORTANCE BASED ON PARTNER

```
partner_yes = df_train_full[df_train_full.partner ==  
'yes'].churn.mean()
```

```
partner_no = df_train_full[df_train_full.partner ==  
'no'].churn.mean()
```

```
partner_yes = df_train_full[df_train_full.partner == 'yes'].churn.mean()  
print('partner == yes:', round(partner_yes, 3))
```

```
partner_no = df_train_full[df_train_full.partner == 'no'].churn.mean()  
print('partner == no :', round(partner_no, 3))
```

```
partner == yes: 0.205  
partner == no : 0.33
```


RISK RATIO

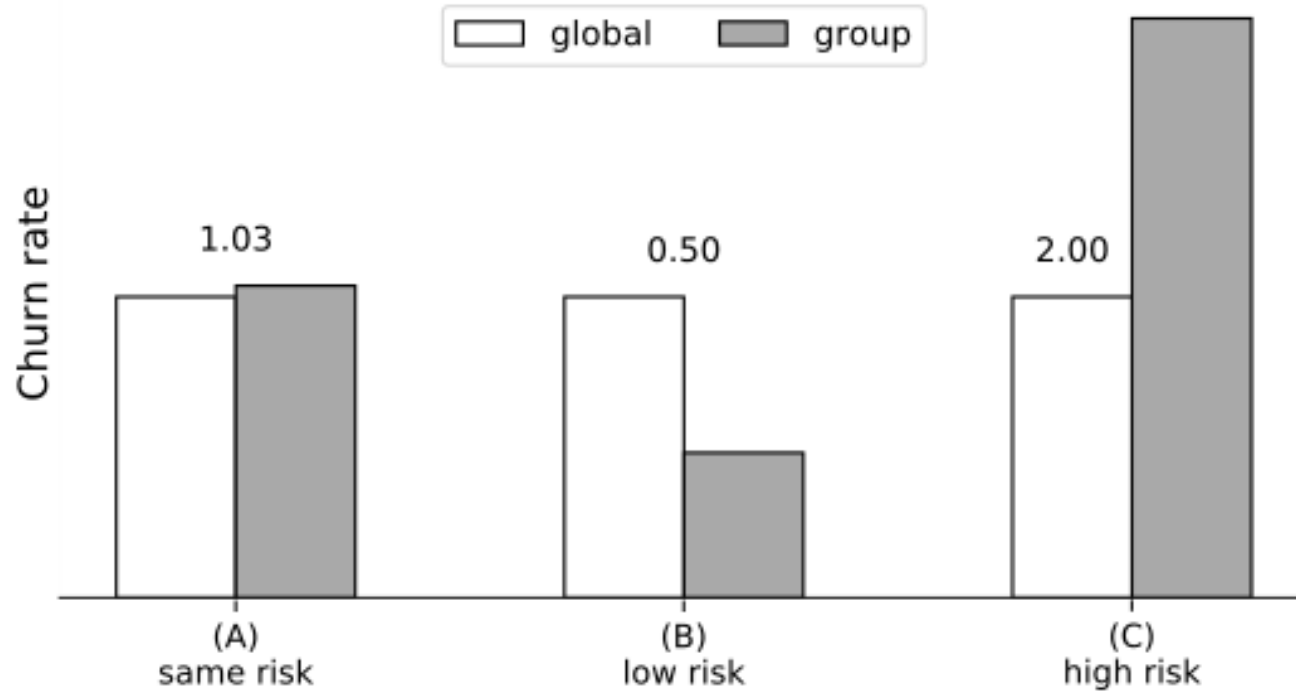
In addition to looking at the difference between the group rate and the global rate, it's interesting to look at the ratio between them. In statistics, the ratio between probabilities in different groups is called risk ratio, where risk refers to the risk of having the effect.

risk = group rate / global rate

For “gender == female”, for example, the risk of churning is 1.02:

risk = 27.7% / 27% = 1.02

RISK RATIO



CONTEXT FOR RISK RATIO

- The term risk originally comes from controlled trials, in which one group of patients is given a treatment (a medicine) and the other group isn't (only a placebo).
- Then we compare how effective the medicine is by calculating the rate of negative outcomes in each group and then calculating the ratio between the rates:

risk = negative outcome rate in group 1 / negative outcome rate in group 2

RISK TABLE

Variable	Value	Churn rate	Risk
Gender	Female	27.7%	1.02
	Male	26.3%	0.97
Partner	Yes	20.5%	0.75
	No	33%	1.22

SQL VERSUS PANDAS (COMPUTE RISK RATIO)

```
SELECT
    gender, AVG(churn),
    AVG(churn) - global_churn,
    AVG(churn) / global_churn
FROM
    data
GROUP BY
    gender
```

```
global_mean = df_train_full.churn.mean()

df_group =
df_train_full.groupby(by='gender').churn.
agg(['mean'])

df_group['diff'] = df_group['mean'] -
global_mean

df_group['risk'] = df_group['mean'] /
global_mean
```

```
df_group
```

	mean	diff	risk
gender			
female	0.276824	0.006856	1.025396
male	0.263214	-0.006755	0.974980

RISK RATIO FOR ALL CATEGORICAL VARIABLES

```
from IPython.display import display

for col in categorical:

    df_group =
df_train_full.groupby(by=col).churn.agg(['mean'])

    df_group['diff'] = df_group['mean'] - global_mean
    df_group['rate'] = df_group['mean'] / global_mean
    display(df_group)
```

ANALYSIS BASED ON RISK RATIO

	mean	diff	risk
gender			
female	0.276824	0.006856	1.025396
male	0.263214	-0.006755	0.974980

(A) Churn ratio and risk: gender

	mean	diff	risk
seniorcitizen			
0	0.242270	-0.027698	0.897403
1	0.413377	0.143409	1.531208

(B) Churn ratio and risk: senior citizen

	mean	diff	risk
partner			
no	0.329809	0.059841	1.221659
yes	0.205033	-0.064935	0.759472

(C) Churn ratio and risk: partner

	mean	diff	risk
phoneservice			
no	0.241316	-0.028652	0.893870
yes	0.273049	0.003081	1.011412

(D) Churn ratio and risk: phone service

- **For gender, there is not much difference between females and males.** Both means are approximately the same, and for both groups the risks are close to 1.
- **Senior citizens tend to churn more than nonseniors:** the risk of churning is 1.53 for seniors and 0.89 for nonseniors.
- **People with a partner churn less than people with no partner.** The risks are 0.75 and 1.22, respectively.
- **People who use phone service are not at risk of churning:** the risk is close to 1, and there's almost no difference with the global churn rate. People who don't use phone service are even less likely to churn: the risk is below 1, and the difference with the global churn rate is negative.

CHURN ANALYSIS

	mean	diff	risk
techsupport			
no	0.418914	0.148946	1.551717
no_internet_service	0.077805	-0.192163	0.288201
yes	0.159926	-0.110042	0.592390

(A) Churn ratio and risk: tech support

	mean	diff	risk
contract			
month-to-month	0.431701	0.161733	1.599082
one_year	0.120573	-0.149395	0.446621
two_year	0.028274	-0.241694	0.104730

(B) Churn ratio and risk: contract

- Clients with no tech support tend to churn more than those who do.
- People with monthly contracts cancel the contract a lot more often than others, and people with two-year contacts churn very rarely.

MUTUAL INFORMATION : CATEGORICAL

“Customers with month-to-month contracts tend to churn a lot more than customers with other kinds of contracts. This is exactly the kind of relationship we want to find in our data. Without such relationships in data, machine learning models will not work – they will not be able to make predictions. The higher the degree of dependency, the more useful a feature is.”

MUTUAL INFORMATION : CATEGORICAL

```
from sklearn.metrics import mutual_info_score

def calculate_mi(series):
    return mutual_info_score(series, df_train_full.churn)

df_mi = df_train_full[categorical].apply(calculate_mi)
df_mi = df_mi.sort_values(ascending=False).to_frame(name='MI')
df_mi
```

MUTUAL INFORMATION : CATEGORICAL

	MI
contract	0.098320
onlinesecurity	0.063085
techsupport	0.061032
internetservice	0.055868
onlinebackup	0.046923

(A) The most useful features according to the mutual information score.

	MI
partner	0.009968
seniorcitizen	0.009410
multiplelines	0.000857
phoneservice	0.000229
gender	0.000117

(B) The least useful features according to the mutual information score.

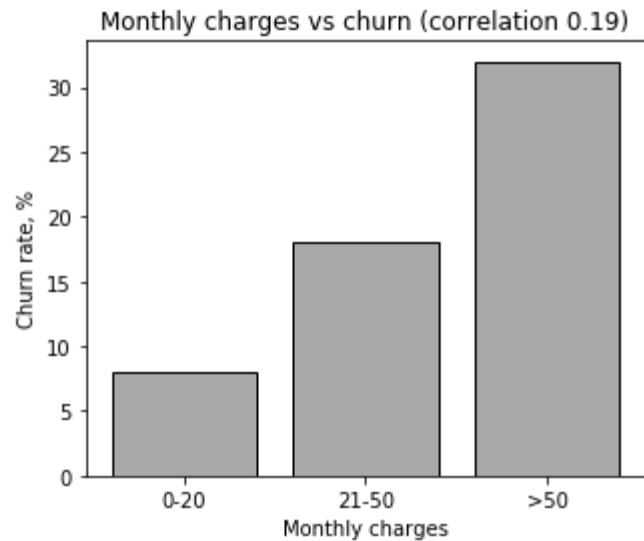
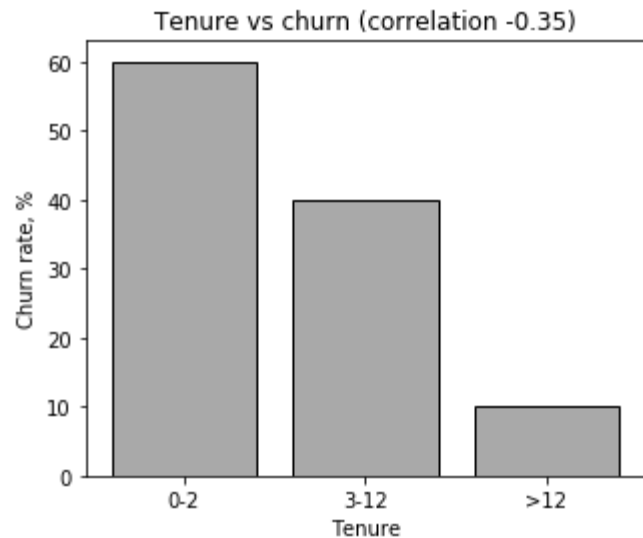
CORRELATION COEFFICIENT

```
df_train_full[numerical].corrwith(df_train_full.churn)
```

Mutual Information shows the degree of dependency of Categorical Variables to the Target Variable.

Correlation does the same with Numeric Variables.

CORRELATION



correlation

tenure -0.351885

monthlycharges 0.196805

totalcharges -0.196353

FEATURE ENGINEERING

Transform all categorical
variables to numeric forms

ONE HOT ENCODING

ONE HOT ENCODING CONCEPT

gender	contract
male	monthly
female	yearly

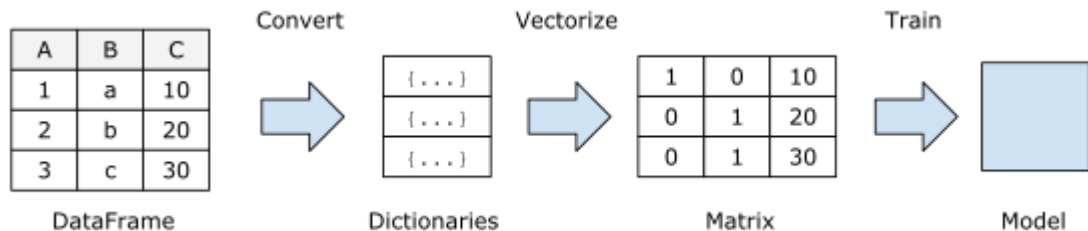


gender		contract		
female	male	monthly	yearly	2-year
0	1	1	0	0
1	0	0	1	0

DICTVECTORIZER

- To use it, we need to convert our dataframe to a list of dictionaries. It's very simple to do in Pandas. Use the `to_dict` method with the `orient="records"` parameter:

```
train_dict = df_train[categorical +  
numerical].to_dict(orient='records')
```



```
{'gender': 'male',  
  
  'seniorcitizen': 0,  
  
  'partner': 'yes',  
  
  'dependents': 'yes',  
  
  'phoneservice': 'yes',  
  
  'multiplelines': 'no',  
  
  'internetservice': 'no',  
  
  'onlinesecurity': 'no_internet_service',  
  
  'onlinebackup': 'no_internet_service',  
  
  'deviceprotection': 'no_internet_service',  
  
  'techsupport': 'no_internet_service',  
  
  'streamingtv': 'no_internet_service',  
  
  'streamingmovies': 'no_internet_service',  
  
  'contract': 'two_year',  
  
  'paperlessbilling': 'no',  
  
  'paymentmethod': 'mailed_check',  
  
  'tenure': 12,  
  
  'monthlycharges': 19.7,  
  
  'totalcharges': 258.35}
```

DICTIONARY VECTORIZER

```
from sklearn.feature_extraction import DictVectorizer
```

```
dv = DictVectorizer(sparse=False)
```

```
dv.fit(train_dict)
```

```
X_train = dv.transform(train_dict)
```

PEEK AT THE VECTORIZED DATA

```
X_train[0]
```

```
dv.get_feature_names()
```

EXERCISE

#How would DictVectorizer encode the following list of

#dictionaries:

```
records = [  
    {'total_charges': 10, 'paperless_billing': 'yes'},  
    {'total_charges': 30, 'paperless_billing': 'no'},  
    {'total_charges': 20, 'paperless_billing': 'no'}  
]
```

MACHINE LEARNING

Predictive Analytics from
the clean Telco Dataset

ML for Classification

LOGISTIC REGRESSION

LINEAR REGRESSION

$$g(x_i) = w_0 + x_i^T w$$

where

x_i is the feature vector corresponding to the i th observation,

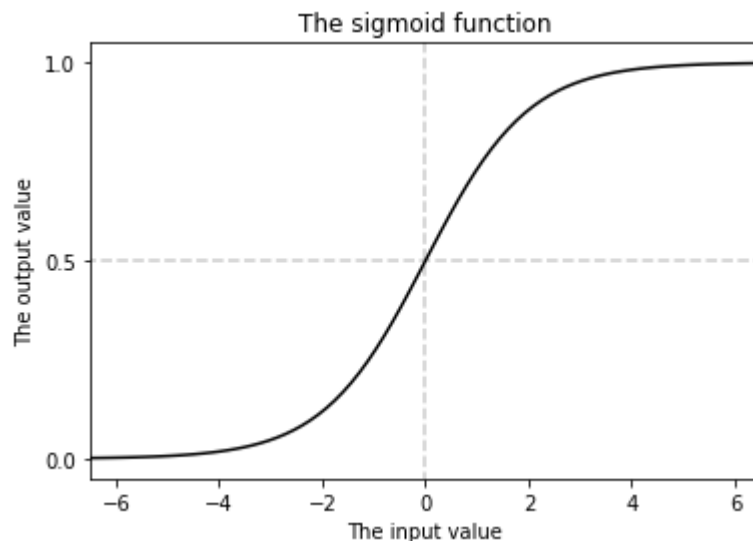
w_0 is the bias term,

w is a vector with the weights of the model.

LOGISTIC REGRESSION

$$g(x_i) = \text{sigmoid}(w_0 + x_i^T w)$$

$$\text{sigmoid}(x) = \frac{1}{1 + \exp(-x)}$$



LINEAR REGRESSION FROM SCRATCH IN PYTHON

```
def linear_regression(xi):  
    result = bias  
    for j in range(n):  
        result = result + xi[j] * w[j]  
    return result
```

LOGISTIC REGRESSION FROM SCRATCH USING PYTHON

```
import math

def logistic_regression(xi):
    score = bias
    for j in range(n):
        score = score + xi[j] * w[j]
    prob = sigmoid(score)
    return prob

def sigmoid(score):
    return 1 / (1 + math.exp(-score))
```

EXERCISE

Why do we need the Sigmoid Function for Logistic Regression?

TRAINING THE LOGISTIC REGRESSION MODEL

TRAINING THE MODEL

```
from sklearn.linear_model import LogisticRegression  
model = LogisticRegression(solver='liblinear', random_state=1)  
model.fit(X_train, y_train)
```

ONE HOT ENCODING

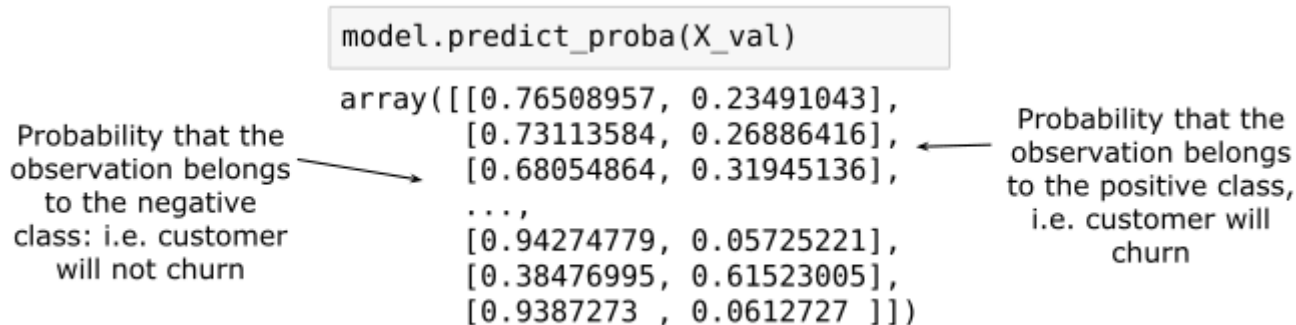
```
val_dict = df_val[categorical +  
numerical].to_dict(orient='records')
```

```
X_val = dv.transform(val_dict)
```

```
y_pred = model.predict_proba(X_val)
```

UNDERSTANDING THE PREDICTIONS



- The predictions of the model: a two-column matrix.
- The first column is the probability that the target=0
- The second column is the probability that the target=1



BUT WE ONLY NEED ONE COLUMN

```
y_pred = model.predict_proba(X_val)[: , 1]
```

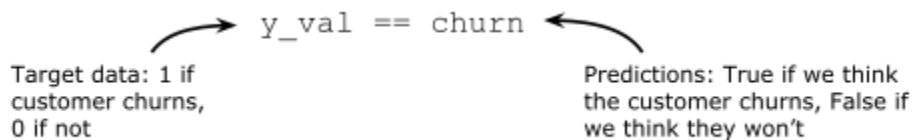
```
y_pred >= 0.5
```

y_pred								
0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
				>= 0.5				
False	False	False	False	True	True	True	True	True

INTRODUCING ACCURACY

```
churn = y_pred >= 0.5
```

```
(y_val == churn).mean() #Quality Measure called ACCURACY
```



ACCURACY

y_val

0	1	0	0	1
---	---	---	---	---

==

churn

False	True	True	False	True
0	1	1	0	1



Cast to integer

ACCURACY

```
(y_val == churn).mean()
```



Boolean array is cast to
integer array



Computing its mean

MODEL INTERPRETATION

UNDERSTAND WHAT YOU'VE DONE : COEFFICIENTS

w_0 is the bias term.

$w = (w_1, w_2, \dots, w_n)$ is the weights vector

- We can get the bias term from `model.intercept_[0]`. When we train our model on all features, the bias term is -0.12
- The rest of the weights are stored in `model.coef_[0]`

```
dict(zip(dv.get_feature_names(), model.coef_[0].round(3)))
```

WEIGHTS

```
{'contract=month-to-month': 0.563,  
  'contract=one_year': -0.086,  
  'contract=two_year': -0.599,  
  'dependents=no': -0.03,  
  'dependents=yes': -0.092,  
  ... # the rest of the weights is omitted  
  'tenure': -0.069,  
  'totalcharges': 0.0}
```

PREPARE A SMALL SUBSET TO BREAK DOWN THE CATEGORICALS

```
small_subset = ['contract', 'tenure', 'totalcharges']
```

```
train_dict_small =
```

```
df_train[small_subset].to_dict(orient='records')
```

```
dv_small = DictVectorizer(sparse=False)
```

```
dv_small.fit(train_dict_small)
```

```
X_small_train = dv_small.transform(train_dict_small)
```

```
dv_small.get_feature_names() 1 ['contract=month-to-month',  
2 'contract=one_year',  
3 'contract=two_year',  
4 'tenure',  
5 'totalcharges']
```


TRAIN THE SMALL SUBSET

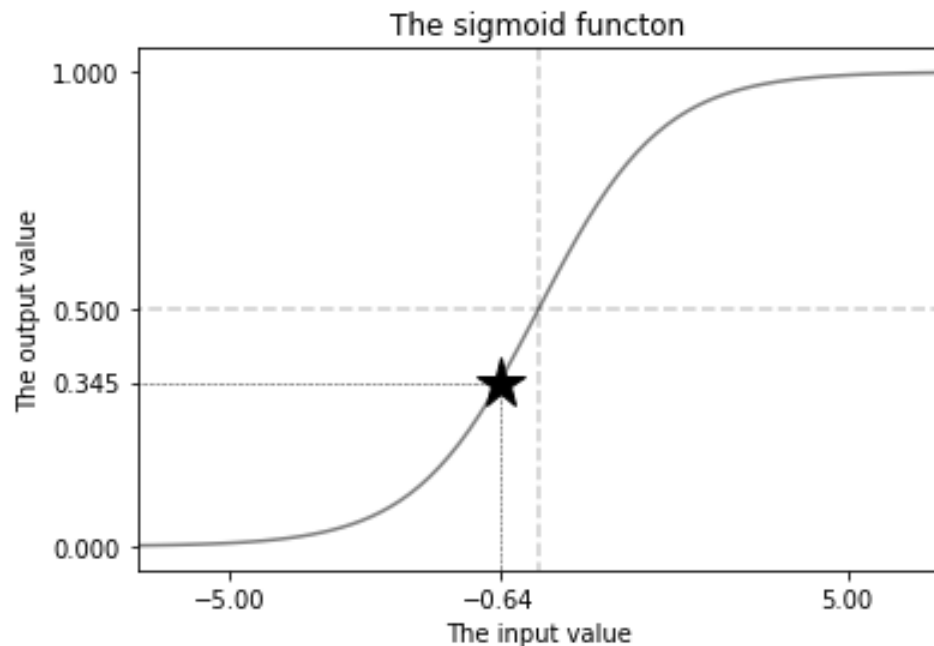
```
model_small = LogisticRegression(solver='liblinear',  
random_state=1)  
  
model_small.fit(X_small_train, y_train)  
  
model_small.intercept_[0] #Check the bias  
  
dict(zip(dv_small.get_feature_names(),  
model_small.coef_[0].round(3))) #Check the other weights  
1 {'contract=month-to-month': 0.91,  
2  'contract=one_year': -0.144,  
3  'contract=two_year': -1.404,  
4  'tenure': -0.097,  
5  'totalcharges': 0.000}
```

THE BIG PICTURE WITH THE CATEGORICALS

bias	contract			tenure	charges
	month	year	2-year		
w0	w1	w2	w3	w4	w5
-0.639	0.91	-0.144	-1.404	-0.097	0.0

THE SIGMOID FUNCTION

The bias term -0.639 on the sigmoid curve. The resulting probability is less than 0.5 so the average customer is more likely not to churn.



UNDERSTANDING THE IMPORTANCE OF CATEGORIES

```
dict(zip(dv_small.get_feature_names(),  
model_small.coef_[0].round(3)))
```

```
'contract=month-to-month': 0.91,
```

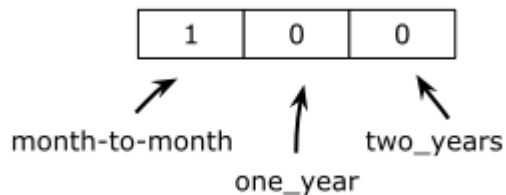
```
'contract=one_year': -0.144,
```

```
'contract=two_year': -1.404.
```

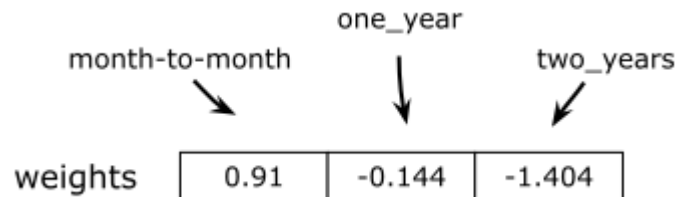
```
1 {'contract=month-to-month': 0.91,  
2  'contract=one_year': -0.144,  
3  'contract=two_year': -1.404,  
4  'tenure': -0.097,  
5  'totalcharges': 0.000}
```

BUILD YOUR INTUITION

The one-hot encoded representation for a customer with a month to month contract



The weights of the month to month, 1 year, 2 year contract features



MAKING A PREDICTION WITH ONE HOT ENCODED CATEGORICAL DATA

- The dot product between the one-hot encoding representation of the contract variable and the corresponding weights.
- The result is 0.91 which is the weight of the hot feature.

$$\begin{array}{rcc} & \text{month-to-month} & \text{one_year} & \text{two_years} \\ \text{contract} & \boxed{1} & \boxed{0} & \boxed{0} \\ & \times & & \\ \text{weights} & \boxed{0.91} & \boxed{-0.144} & \boxed{-1.404} \\ & = & & \\ & \boxed{1} \cdot \boxed{0.91} & + & \boxed{0} \cdot \boxed{-0.144} & + & \boxed{0} \cdot \boxed{-1.404} \\ & \text{month-to-month} & & \text{one_year} & & \text{two_years} \\ & = & & & & \\ & 0.91 & & & & \end{array}$$

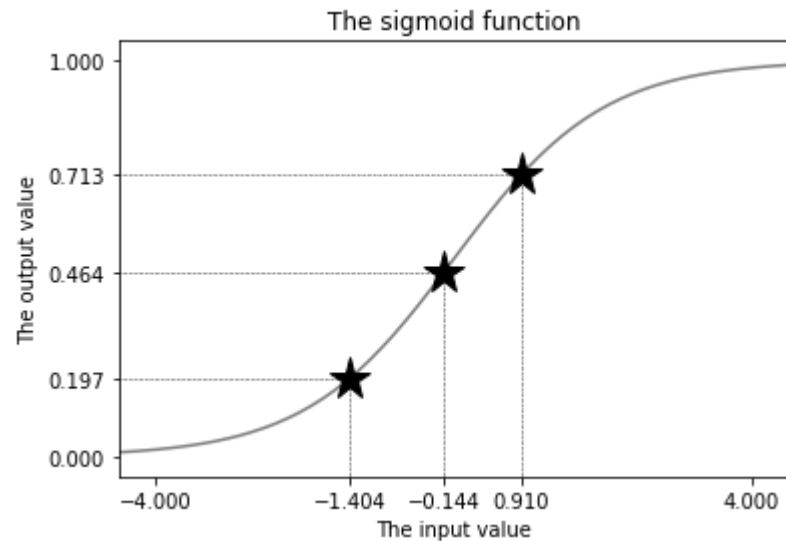
SAME THING BUT A 2 YEAR CONTRACT CLIENT

Only the HOT feature is ever factored into the model.

$$\begin{array}{rcccl} & \text{month-to-month} & & \text{one_year} & & \text{two_years} \\ & \swarrow & & \downarrow & & \swarrow \\ \text{contract} & \boxed{0} & \boxed{0} & \boxed{1} \\ & \times & & & & \\ \text{weights} & \boxed{0.91} & \boxed{-0.144} & \boxed{-1.404} \\ & = & & & & \\ & \boxed{0} \cdot \boxed{0.91} & + & \boxed{0} \cdot \boxed{-0.144} & + & \boxed{1} \cdot \boxed{-1.404} \\ & \text{month-to-month} & & \text{one_year} & & \text{two_years} \\ & = & & & & \\ & -1.404 & & & & \end{array}$$

DOES THE SIGN (+/-) HAVE MEANING?

The sign of the weight matters. If it is positive - churn.
Negative - loyal customer.



LET'S ANALYZE THE NUMERICAL FEATURES

The score model calculate for a customer with a m2m contract, 12 months of tenure and total charges of \$1,000

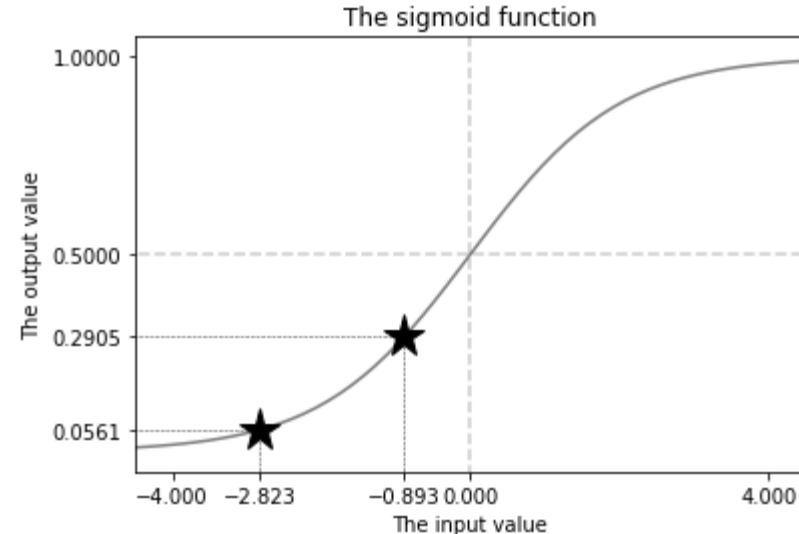
$$-0.639 + 0.91 - 12 \cdot 0.097 + 0 \cdot 1000 = -0.893$$

bias	monthly contract	12 months of tenure	total charges don't matter	negative, so low likelihood of churn
------	------------------	---------------------	----------------------------	--------------------------------------

The score model calculate for a customer with a yearly contract, 24 months of tenure and total charges of \$2,000

$$-0.639 + 0.144 - 24 \cdot 0.097 + 0 \cdot 2000 = -2.823$$

bias	yearly contract	24 months of tenure	total charges don't matter	negative, very low likelihood of churn
------	-----------------	---------------------	----------------------------	--



USING THE CHURN MODEL

```
customer = {
```

```
    'customerid': '8879-zkjof',
```

```
    'gender': 'female',
```

```
    'seniorcitizen': 0,
```

```
    'partner': 'no',
```

```
    'dependents': 'no',
```

```
    'tenure': 41,
```

```
    'phoneservice': 'yes',
```

```
    'multiplelines': 'no',
```

```
    'internetservice': 'dsl',
```

```
    'onlinesecurity': 'yes',
```

```
    'onlinebackup': 'no',
```

```
#Continuation
```

```
    'deviceprotection': 'yes',
```

```
        'techsupport': 'yes',
```

```
        'streamingtv': 'yes',
```

```
        'streamingmovies': 'yes',
```

```
        'contract': 'one_year',
```

```
        'paperlessbilling': 'yes',
```

```
        'paymentmethod': 'bank_transfer_(a
```

```
        'monthlycharges': 79.85,
```

```
        'totalcharges': 3320.75,
```

```
}
```

VECTORIZED INPUT

```
X_test = dv.transform([customer])
```

```
#Output
```

```
[[ 0.  ,  1.  ,  0.  ,  1.  ,  0.  ,  0.  ,  0.  ,  
   1.  ,  1.  ,  0.  ,  1.  ,  0.  ,  0.  , 79.85,  
   1.  ,  0.  ,  0.  ,  1.  ,  0.  ,  0.  ,  0.  ,  
   0.  ,  1.  ,  0.  ,  1.  ,  1.  ,  0.  ,  1.  ,  
   0.  ,  0.  ,  0.  ,  0.  ,  1.  ,  0.  ,  0.  ,  
   0.  ,  1.  ,  0.  ,  0.  ,  1.  ,  0.  ,  0.  ,  
   1.  , 41.  , 3320.75]]
```

PUT THE MATRIX INTO THE TRAINED MODEL

```
model.predict_proba(X_test)
```

```
#output
```

```
[[0.93, 0.07]]
```

```
X_test = dv.transform([customer])  
model.predict_proba(X_test)[0, 1]
```

```
0.07332577315357781
```

All we need from the matrix is the number at the first row and second column: the probability of churning for this customer. To select this number from the array, we use the brackets operator:

```
model.predict_proba(X_test)[0, 1]
```

TAKE A LOOK AT ANOTHER CUSTOMER

```
customer = {  
    'gender': 'female',  
    'seniorcitizen': 1,  
    'partner': 'no',  
    'dependents': 'no',  
    'phoneservice': 'yes',  
    'multiplelines': 'yes',  
    'internetservice': 'fiber_optic',  
    'onlinesecurity': 'no',  
    'onlinebackup': 'no',  
    'deviceprotection': 'no',  
    'techsupport': 'no',  
    'streamingtv': 'yes',  
    'streamingmovies': 'no',  
    'contract': 'month-to-month',  
    'paperlessbilling': 'yes',  
    'paymentmethod': 'electronic_check',  
    'tenure': 1,  
    'monthlycharges': 85.7,  
    'totalcharges': 85.7  
}
```

LET'S MAKE A PREDICTION

```
X_test = dv.transform([customer])
```

```
model.predict_proba(X_test)[0, 1]
```

```
In [62]: X_test = dv.transform([customer])  
         model.predict_proba(X_test)[0, 1]
```

```
Out[62]: 0.8321638622459152
```


PROJECT

- Classification models are often used for marketing purposes, and one of the problems it solves is lead scoring.
- A lead is a potential customer who may convert (became an actual customer) or not.
- In this case, the conversion is the target we want to predict.
- You can take a dataset from **<https://www.kaggle.com/ashydv/leads-dataset>** and build a model for that.
- You may notice that the lead scoring problem is very similar to churn prediction, but in one case we want to get a new client to sign a contract with us, and in another case we want a client not to cancel the contract.

PROJECT 2

- Another popular application of classification is default prediction, which is estimating the risk of a customer's not returning a loan.
- In this case, the variable we want to predict is default, and it also has two outcomes: whether the customer managed to pay back the loan in time (good customer) or not (default).
- There are many datasets online that you can use for training a model, such as <https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients> (or, the same one available via kaggle: <https://www.kaggle.com/pratjain/credit-card-default>).

SUMMARY

Classification