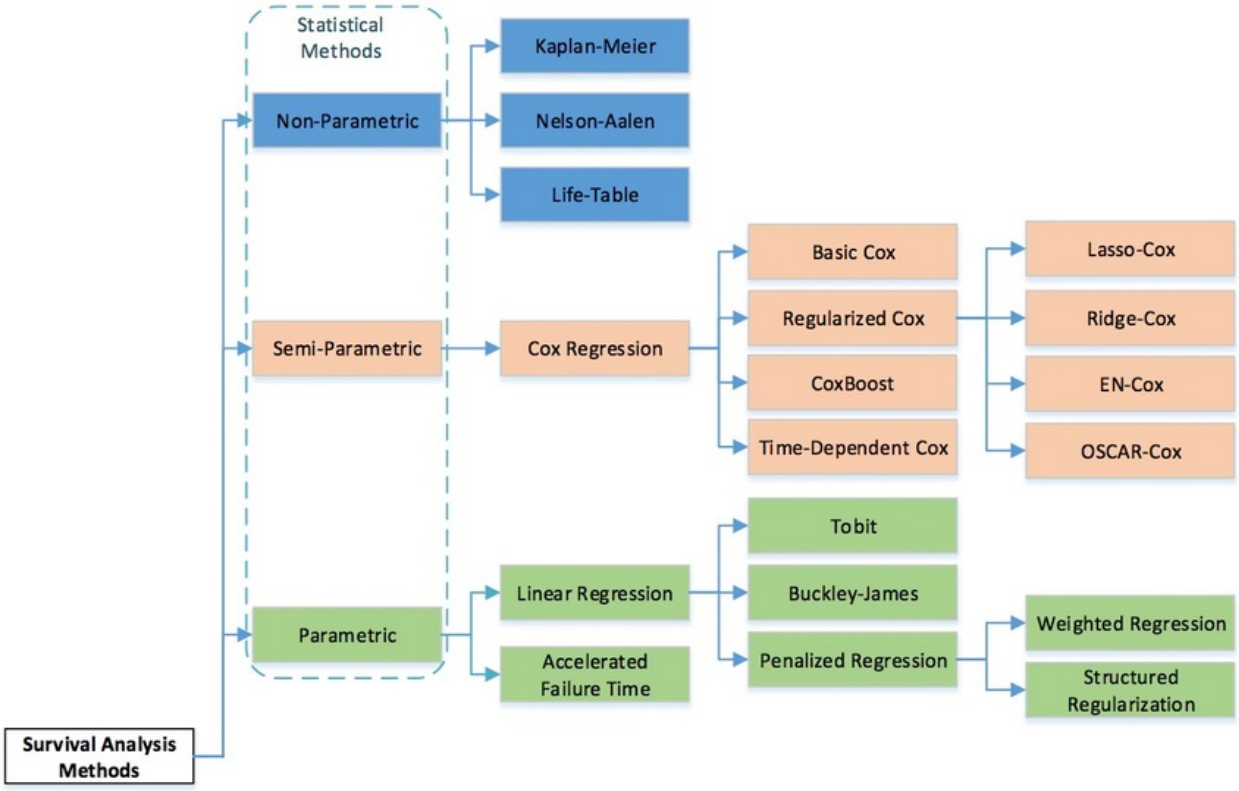
Problem Statement:

1) To predict if the price will change with the next search hit
2) When will the next search hit come?

Methodology:

For the 1st problem statement, we have performed Survival Analysis for predicting if our event (price change) will occur, given some covariates.



Overview of methods available in Survival Analysis

Data Preparation

To be fully compatible with scikit-learn, we need $Status$ (whether the actual survival time was observed or if was censored) and $Survival\_in\_days$ (the observed survival time) needs to be stored as a structured array.

We create a *cumulative price lag* column ($Survival\_in\_days$) and a boolean column which records non-zero price change in consecutive searches ($Status$). We also break our search timestamp into elementary components to build multivariate features.

	timeStamp	departDate	totalFare	sector	advPurchaseDays	pricediff	timediff	change	survival_mins	year	month	date	weekday	hour	minute
0	2022-01-19 00:29:41	2022-01-20	7424	DEL-GOI	0	0.0	27.800000	False	27.800000	2022	1	19	2	0	29
1	2022-01-19 01:02:34	2022-01-20	7424	DEL-GOI	0	0.0	32.883333	False	60.683333	2022	1	19	2	1	2
2	2022-01-19 01:30:18	2022-01-20	14319	DEL-GOI	0	6895.0	27.733333	True	88.416667	2022	1	19	2	1	30
3	2022-01-19 02:32:44	2022-01-20	14319	DEL-GOI	0	0.0	62.433333	False	150.850000	2022	1	19	2	2	32
4	2022-01-19 02:48:43	2022-01-20	7424	DEL-GOI	0	-6895.0	15.983333	True	166.833333	2022	1	19	2	2	48
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1416	2022-02-08 20:31:57	2022-02-08	17368	DEL-GOI	-1	9944.0	31.850000	True	31.850000	2022	2	8	1	20	31
1417	2022-02-08 20:46:40	2022-02-08	7424	DEL-GOI	-1	-9944.0	14.716667	True	14.716667	2022	2	8	1	20	46
1418	2022-02-08 21:01:01	2022-02-08	7424	DEL-GOI	-1	0.0	14.350000	False	29.066667	2022	2	8	1	21	1
1419	2022-02-08 21:22:23	2022-02-08	7424	DEL-GOI	-1	0.0	21.366667	False	50.433333	2022	2	8	1	21	22
1420	2022-02-08 21:25:40	2022-02-08	23953	DEL-GOI	-1	16529.0	3.283333	True	53.716667	2022	2	8	1	21	25

So far, we have tried out Kaplan Meier Estimates (for univariate data), Cox Proportional Hazards Survival Analysis, and Random Survival Forests.

Metric Understanding (for model benchmarking)

Kaplan-Meier vs Cox Proportional Hazards

KM Survival Analysis cannot use multiple predictors/features, whereas Cox Regression can.

Brier Score

The time-dependent Brier score is the mean squared error at time point *t*

$$BS = \frac{1}{N}\sum_{t=1}^{N}(f_t - o_t)^2$$

in which f_{t} is the probability that was forecast, o_{t} the actual outcome of the event at instance t (0 if it does not happen and 1 if it does happen) and N is the number of forecasting instances.
The value of the Brier score is always between 0.0 and 1.0, where a model with perfect skill has a score of 0.0 and the worst has a score of 1.0
Integrated Brier Score is B.S. for all time t.

Concordance Index

Proportion of all concordant comparable pairs (ordering_)
CI = 1, pairs are perfectly ordered
CI = 0, all pairs are incorrectly ordered

Concordance Index IPCW (inverse probability of censoring weights)

Does not depend on the distribution of censoring times in the test data, unbiased and consistent for a population concordance measure that is free of censoring.
Inversely weight regression analyses by the probability of participation effectively inflates the impact of underrepresented subjects, so we can observe associations that would have been observed if all subjects had stayed in the study, assuming the models are correctly specified.

Cumulative/Dynamic AUC

The associated cumulative/dynamic AUC quantifies how well a model can distinguish subjects who fail by a given time ***(ti≤t)*** from subjects who fail after this time ***(ti>t)***.
Gives us a model's ROC-AUC at different times.

Model Integrated Metrics (Random Survival Forest)

1. Brier Score
2. Cumulative/Dynamic AUC
3. Concordance Index IPCW

```
asbs = as_integrated_brier_score_scorer(rsf, times)
asas = as_cumulative_dynamic_auc_scorer(rsf, times)
asis = as_concordance_index_ipcw_scorer(rsf)

asbs.fit(X_train, y_train)
asas.fit(X_train, y_train)
asis.fit(X_train, y_train)

print('asbs', asbs.score(X_test, y_test))
print('asas', asas.score(X_test, y_test))
print('asis', asis.score(X_test, y_test))
```

```
asbs -0.04438544711740865
asas 0.7582115993783259
asis 0.6662956698739918
```

Observations:

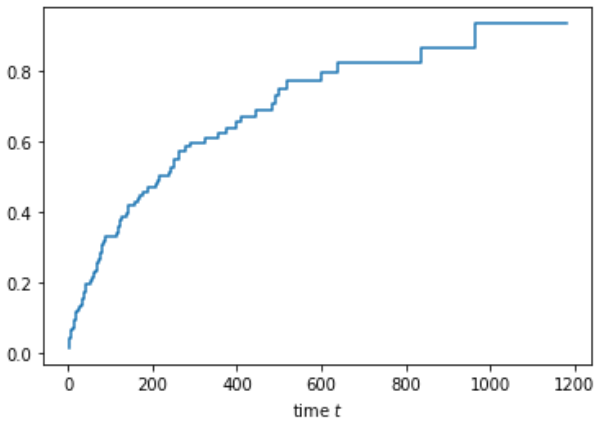
Brier Score (model integrated scores are negative values only) and AUC Scores are very good, C-Index is not that good.

Building a custom metric for benchmarking output performance

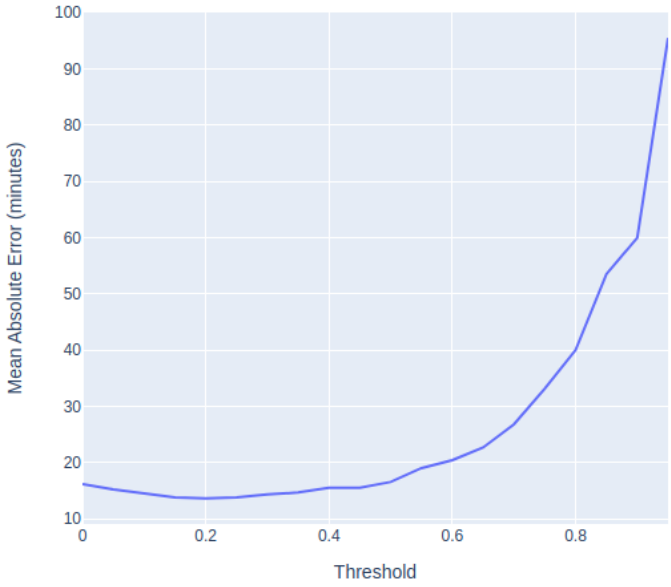
By definition, the survival function $S(t) = P(T > t)$ is the probability of a patient's survival after time t.

Analogously, for our case, it represents probability that the price change will not occur till time t.

So, we create a *death function* $D(t) = 1 - S(t)$ to describe the probability of price change after time t. A death function looks like this:



Now after obtaining death functions for all patients (search hits), we move on to selecting a probability threshold τ where the error between *the actual time of price change* and the *time corresponding to the threshold probability from the death function* is minimum.



Note: This threshold τ is obtained from the training data.

Based on the threshold τ selected, we build predictions for the test data and calculate error between the ground truth and the predictions.