Documentation

The following code Identifies differentially expressed genes (DEGs) associated with overall survival (OS) in stage I and III breast cancer patients using Cox proportional hazards regression with L1/L2 regularization.

R Part (Abbreviated)

- Download from the TCGAbiolink the survival data adapted from cbioportal related to the patients with no H.pylori infection.

```r
mrna_query <- GDCquery(project = "TCGA-STAD",
                 data.category = "Transcriptome Profiling",
                 data.type = "Gene Expression Quantification",
                        workflow.type = "STAR - Counts",
                       experimental.strategy = "RNA-Seq",
                        barcode=mrna_meta$Sample.ID) #adapted from cbioportals

GDCdownload(mrna_query,directory=
                      "T:\\Programming\\Data\\TMN Project\\Data\\RNA Seq - STAD")
mrna_df<-GDCprepare(mrna_query, directory=
                      "T:\\Programming\\Data\\TMN Project\\Data\\RNA Seq - STAD")
```

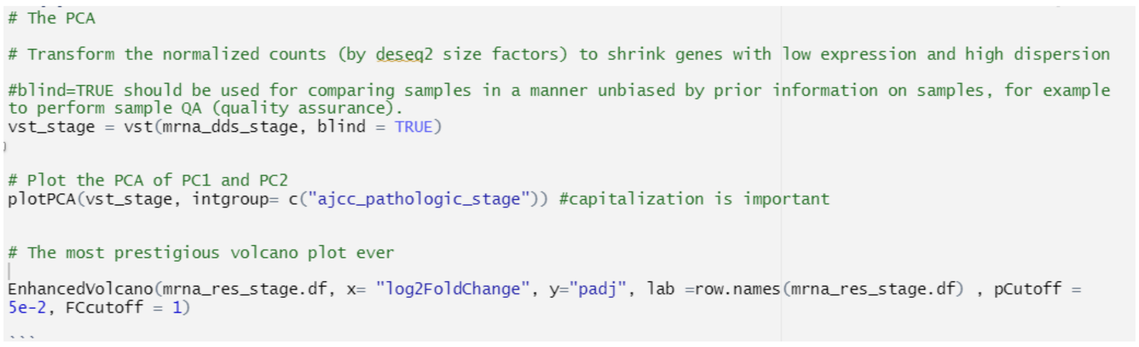
- Patients Selection (Stage I vs. III was used instead of Stage I vs. IV, as stage IV has significantly low number of samples(14))

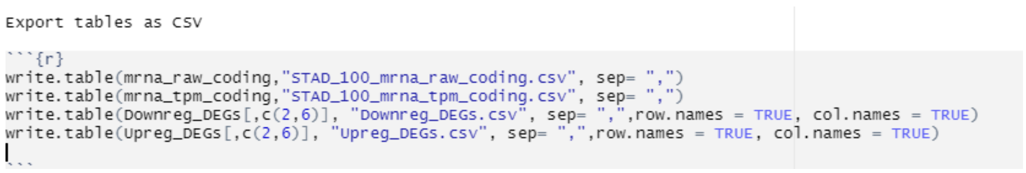
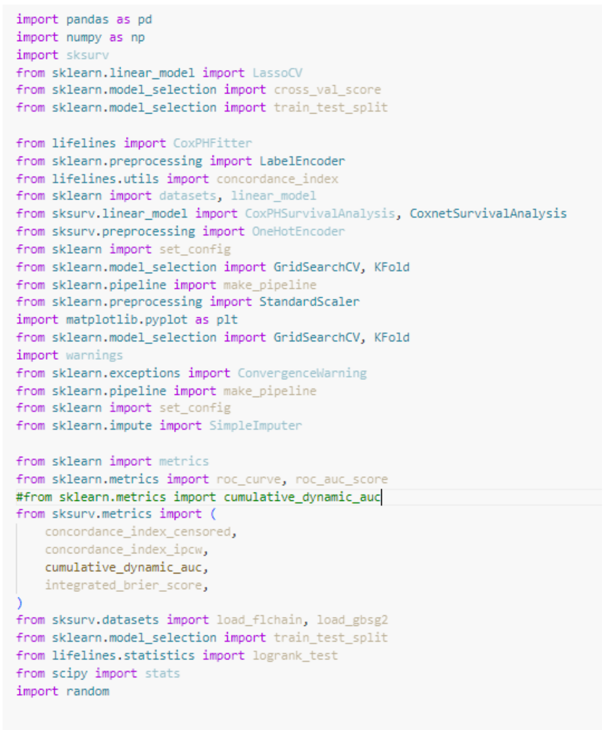
```r
mrna_meta=as.data.frame((colData(mrna_df)),
    options = list(scrollX = TRUE, keys = TRUE, pageLength = 5),
    rownames = FALSE)

mrna_meta$ajcc_pathologic_stage <- str_remove_all(mrna_meta$ajcc_pathologic_stage, "[A-C]")

#Choosing only stage I and III or IV
mrna_meta= mrna_meta[mrna_meta$ajcc_pathologic_stage == "Stage I" |mrna_meta$ajcc_pathologic_stage == "Stage III",]
#mrna_meta= mrna_meta[mrna_meta$ajcc_pathologic_stage == "Stage I" |mrna_meta$ajcc_pathologic_stage == "Stage IV",] # Low number
of stage IV samples (14) and lower number of DEGs (76 in comparison to 274)
mrna_meta= mrna_meta[!is.na(mrna_meta$ajcc_pathologic_stage),]
```

- Deseq2 was applied on StageI and III patients on the Raw transcriptomic data

```r
## DESeq2 Analysis Macrophage
mrna_dds_stage <- DESeqDataSetFromMatrix(
  countData=round(mrna_raw_coding),
  colData = mrna_meta,
  design = ~ ajcc_pathologic_stage)

mrna_dds_stage$ajcc_pathologic_stage <- relevel(mrna_dds_stage$ajcc_pathologic_stage, ref = "Stage
I")
mrna_dds_stage = DESeq(mrna_dds_stage)

#Why do you run the below code (line240)?

#This normalized count line will be later needed for the heatmap

normCounts <- counts(mrna_dds_stage, normalized=TRUE)

## DESeq2 results
mrna_res_stage=results(mrna_dds_stage , alpha = 0.05, lfcThreshold = 0, contrast =
c("ajcc_pathologic_stage", "Stage III", "Stage I"))
summary(mrna_res_stage)
mrna_res_stage.df= as.data.frame(mrna_res_stage)
```

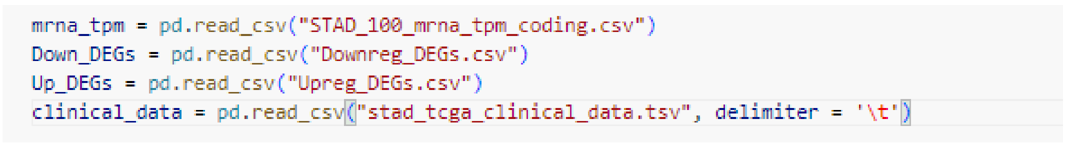
- Datavisualization (PCA and Volcano Plot)



```
# The PCA

# Transform the normalized counts (by deseq2 size factors) to shrink genes with low expression and high dispersion

#blind=TRUE should be used for comparing samples in a manner unbiased by prior information on samples, for example
to perform sample QA (quality assurance).
vst_stage = vst(mrna_dds_stage, blind = TRUE)


# Plot the PCA of PC1 and PC2
plotPCA(vst_stage, intgroup= c("ajcc_pathologic_stage")) #capitalization is important


# The most prestigious volcano plot ever

EnhancedVolcano(mrna_res_stage.df, x= "log2FoldChange", y="padj", lab =row.names(mrna_res_stage.df) , pCutoff =
5e-2, FCcutoff = 1)

```
```

- Data Export



Export tables as CSV

```
```{r}
write.table(mrna_raw_coding,"STAD_100_mrna_raw_coding.csv", sep= ",")
write.table(mrna_tpm_coding,"STAD_100_mrna_tpm_coding.csv", sep= ",")
write.table(Downreg_DEGS[,c(2,6)], "Downreg_DEGS.csv", sep= ",",row.names = TRUE, col.names = TRUE)
write.table(Upreg_DEGS[,c(2,6)], "Upreg_DEGS.csv", sep= ",",row.names = TRUE, col.names = TRUE)

```
```

Python Part (Installation)

First of all, you have to install the following packages in order to functionally clone the code in your environment. The package mainly consists of Sci-kit learn modules for pre-processing and machine learning, lifeline for survival analysis, NumPy and Pandas for data manipulation.



```
import pandas as pd
import numpy as np
import sksurv
from sklearn.linear_model import LassoCV
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split

from lifelines import CoxPHFitter
from sklearn.preprocessing import LabelEncoder
from lifelines.utils import concordance_index
from sklearn import datasets, linear_model
from sksurv.linear_model import CoxPHSurvivalAnalysis, CoxnetSurvivalAnalysis
from sksurv.preprocessing import OneHotEncoder
from sklearn import set_config
from sklearn.model_selection import GridSearchCV, KFold
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
from sklearn.model_selection import GridSearchCV, KFold
import warnings
from sklearn.exceptions import ConvergenceWarning
from sklearn.pipeline import make_pipeline
from sklearn import set_config
from sklearn.impute import SimpleImputer

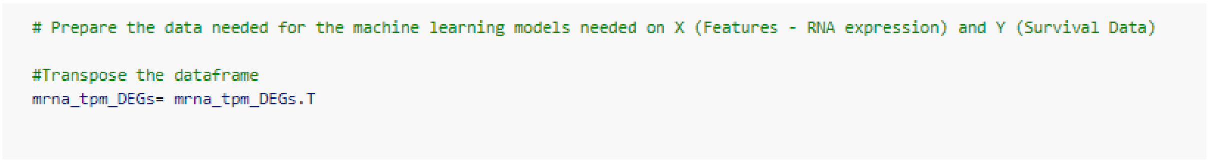
from sklearn import metrics
from sklearn.metrics import roc_curve, roc_auc_score
#from sklearn.metrics import cumulative_dynamic_auc
from sksurv.metrics import (
    concordance_index_censored,
    concordance_index_ipcw,
    cumulative_dynamic_auc,
    integrated_brier_score,
)
from sksurv.datasets import load_flchain, load_gbsg2
from sklearn.model_selection import train_test_split
from lifelines.statistics import logrank_test
from scipy import stats
import random
```

Import of Gene Expression and Survival Data
```



```
mrna_tpm = pd.read_csv("STAD_100_mrna_tpm_coding.csv")
Down_DEGs = pd.read_csv("Downreg_DEGs.csv")
Up_DEGs = pd.read_csv("Upreg_DEGs.csv")
clinical_data = pd.read_csv("stad_tcga_clinical_data.tsv", delimiter = '\t')
```

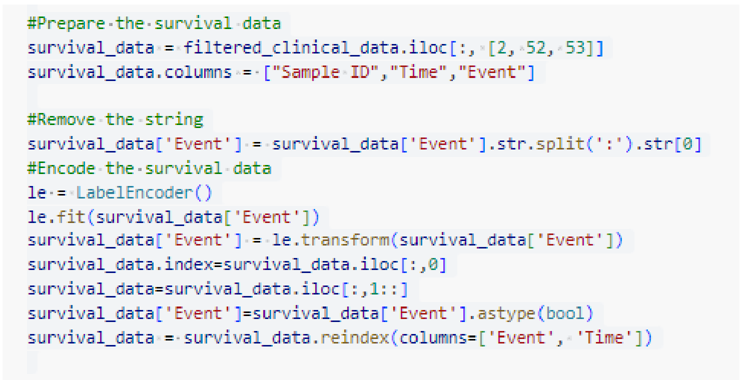
- clinical_data: DataFrame containing patient clinical information, specifically includes the overall survival.
- mrna_tpm: DataFrame containing RNA expression data for the patients with clinical data.
- Down/Up_DEGs: DataFrame containing the DEGs between stage I and III gastric cancer.

Pre-processing of Gene Expression and Survival Data



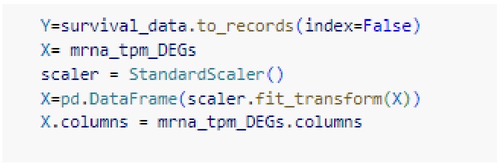
```
# Prepare the data needed for the machine learning models needed on X (Features - RNA expression) and Y (Survival Data)

#Transpose the dataframe
mrna_tpm_DEGs= mrna_tpm_DEGs.T
```

The expression matrix that will be fed into the elasticnet-cox model has to have the features (genes) as columns and samples (patients) as rows. and the only the DEGs were used as features to further test and select from.



```
#Prepare the survival data
survival_data = filtered_clinical_data.iloc[:, [2, 52, 53]]
survival_data.columns = ["Sample ID","Time","Event"]

#Remove the string
survival_data['Event'] = survival_data['Event'].str.split(':').str[0]
#Encode the survival data
le = LabelEncoder()
le.fit(survival_data['Event'])
survival_data['Event'] = le.transform(survival_data['Event'])
survival_data.index=survival_data.iloc[:,0]
survival_data=survival_data.iloc[:,1::]
survival_data['Event']=survival_data['Event'].astype(bool)
survival_data = survival_data.reindex(columns=['Event', 'Time'])
```

For the survival data, only two columns are adapted, which are the (Time) that refers to the time since the beginning of diagnosis till the patient was censored or died, and (Event) which clarify whether the patient censored (0) or lost follow up – or died (1). Moreover, the survival time has to be turned bool (False instead of 0 and True instead of 1)



```
Y=survival_data.to_records(index=False)
X= mrna_tpm_DEGs
scaler = StandardScaler()
X=pd.DataFrame(scaler.fit_transform(X))
X.columns = mrna_tpm_DEGs.columns
```

Importantly, the survival time has to be turned into records instead of a dataframe (False,30) , which stands for a censored event for a patient who lived 30 months after diagnosis. For the gene expression data, a normalized Z-score was applied on the expression values using the StandardScaler() function.

ElasticNet Model and Parameter Selection

```
random.seed(1)

#Testing the range of alphas and their corresponding number of genes selected
estimator=CoxnetSurvivalAnalysis(n_alphas=10, l1_ratio=1, alpha_min_ratio=0.01 ,verbose = 10)
estimator.fit(X,Y)

#Making the dataframe for the coefficients of each genes corresponding to that alpha value
coefficients_lasso=pd.DataFrame(estimator.coef_, index=X.columns, columns=np.round(estimator.alpha
alphas_estimated=estimator.alphas_
alphas=estimator.alphas_

for i, col in enumerate(coefficients_lasso.columns):
    non_zero = np.sum(coefficients_lasso[col] != 0)  # Count non-zeros in each column
    print(f"Column {i+1}: Number of non-zero coefficients: {non_zero}")
```

As you can see in the figure above, the first important aspect in the following chunks is setting a random seed to avoid slight variations in the results. For the CoxnetSurvivalAnalysis() function takes the following as inputs:

- L1: which is the value that determines whether lasso (1) will be used or (0) ridge or (0.1-0.9) a hybrid penalty model will be adapted
- alpha_min_ratio is the different values of alphas that will be tested (penalty strength) starting from the initial or the default alpha set (0.18)

2. Cox Model Fitting and Analysis and Grid Search for Hyperparameter Tuning:

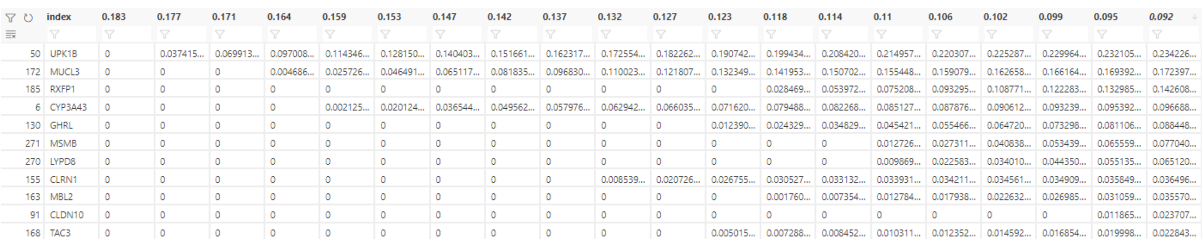
   ○ Fits CoxnetSurvivalAnalysis with L1 regularization (Lasso only/Initialization Step).

```
#Testing the range of alphas and their corresponding number of genes selected
estimator=CoxnetSurvivalAnalysis(n_alphas=10, l1_ratio=1, alpha_min_ratio=0.01 ,verbose = 10)
estimator.fit(X,Y)
```

   ○ The previously fitted model will creates hazard coefficients for the DEGs at different alphas. You can see in the next figure that a dataframe called "Coefficients lasso"– will contain the estimated factors in a matrix, with the excluded features having 0s as their coefficients.

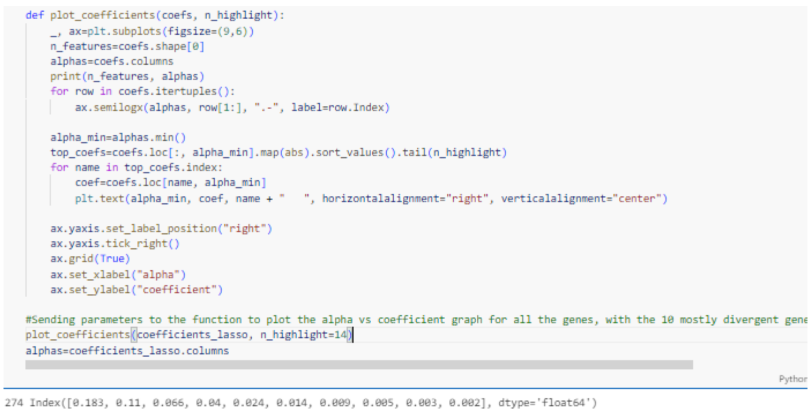
```
coefficients_lasso=pd.DataFrame(estimator.coef_, index=X.columns, columns=np.round(estimator.alphas_, 5))
alphas=estimator.alphas_

for i, col in enumerate(coefficients_lasso.columns):
    non_zero = np.sum(coefficients_lasso[col] != 0)  # Count non-zeros in each column
    print(f"Column {i+1}: Number of non-zero coefficients: {non_zero}")
```

```
Column 1: Number of non-zero coefficients: 0
Column 2: Number of non-zero coefficients: 12
Column 3: Number of non-zero coefficients: 24
Column 4: Number of non-zero coefficients: 40
Column 5: Number of non-zero coefficients: 50
Column 6: Number of non-zero coefficients: 62
Column 7: Number of non-zero coefficients: 65
Column 8: Number of non-zero coefficients: 70
Column 9: Number of non-zero coefficients: 76
Column 10: Number of non-zero coefficients: 76
```



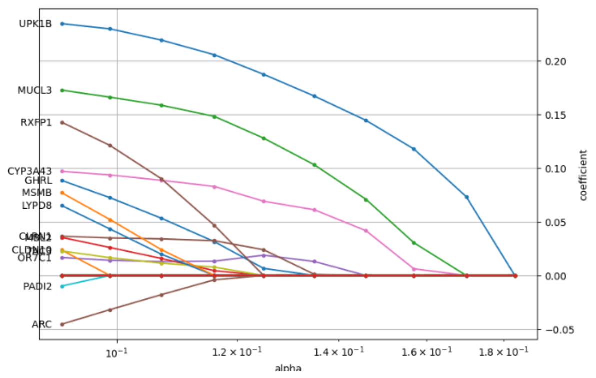
	index	0.183	0.177	0.171	0.164	0.159	0.153	0.147	0.142	0.137	0.132	0.127	0.123	0.118	0.114	0.11	0.106	0.102	0.099	0.095	0.092
50	UPK1B	0	0.037415...	0.069913...	0.097008...	0.114346...	0.128150...	0.140403...	0.151661...	0.162317...	0.172554...	0.182262...	0.190742...	0.199434...	0.208420...	0.214957...	0.220307...	0.225287...	0.229964...	0.232105...	0.234226...
172	MUCL3	0	0	0	0.004686...	0.025726...	0.046491...	0.065117...	0.081835...	0.096830...	0.110023...	0.121807...	0.132349...	0.141953...	0.150702...	0.155448...	0.159079...	0.162658...	0.166164...	0.169392...	0.172397...
185	RXFP1	0	0	0	0	0	0	0	0	0	0	0	0	0.028469...	0.053972...	0.075208...	0.093295...	0.108771...	0.122283...	0.132985...	0.142608...
6	CYP3A43	0	0	0	0	0.002125...	0.020124...	0.036544...	0.049562...	0.057976...	0.062942...	0.066035...	0.071620...	0.079488...	0.082268...	0.085127...	0.087876...	0.090612...	0.093239...	0.095392...	0.096688...
130	GHRL	0	0	0	0	0	0	0	0	0	0	0	0.012390...	0.024329...	0.034829...	0.045421...	0.055466...	0.064720...	0.073298...	0.081106...	0.088448...
271	MSMB	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.012726...	0.027311...	0.040838...	0.053439...	0.065559...	0.077040...
270	LYPD8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.009869...	0.022583...	0.034010...	0.044350...	0.055135...	0.065120...
155	CLRN1	0	0	0	0	0	0	0	0	0	0.008539...	0.020726...	0.026755...	0.030527...	0.033132...	0.033931...	0.034211...	0.034561...	0.034909...	0.035849...	0.036496...
163	MBL2	0	0	0	0	0	0	0	0	0	0	0	0	0.001760...	0.007354...	0.012784...	0.017938...	0.022632...	0.026985...	0.031059...	0.035570...
91	CLDN10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.011865...	0.023707...
168	TAC3	0	0	0	0	0	0	0	0	0	0	0	0.005015...	0.007288...	0.008452...	0.010311...	0.012352...	0.014592...	0.016854...	0.019998...	0.022843...

   ○ Plots coefficients vs. alphas, highlighting important genes.



```python
def plot_coefficients(coefs, n_highlight):
    _, ax=plt.subplots(figsize=(9,6))
    n_features=coefs.shape[0]
    alphas=coefs.columns
    print(n_features, alphas)
    for row in coefs.itertuples():
        ax.semilogx(alphas, row[1:], ".-", label=row.Index)

    alpha_min=alphas.min()
    top_coefs=coefs.loc[:, alpha_min].map(abs).sort_values().tail(n_highlight)
    for name in top_coefs.index:
        coef=coefs.loc[name, alpha_min]
        plt.text(alpha_min, coef, name + "   ", horizontalalignment="right", verticalalignment="center")

    ax.yaxis.set_label_position("right")
    ax.yaxis.tick_right()
    ax.grid(True)
    ax.set_xlabel("alpha")
    ax.set_ylabel("coefficient")

#Sending parameters to the function to plot the alpha vs coefficient graph for all the genes, with the 10 mostly divergent gene
plot_coefficients(coefficients_lasso, n_highlight=14)
alphas=coefficients_lasso.columns
```

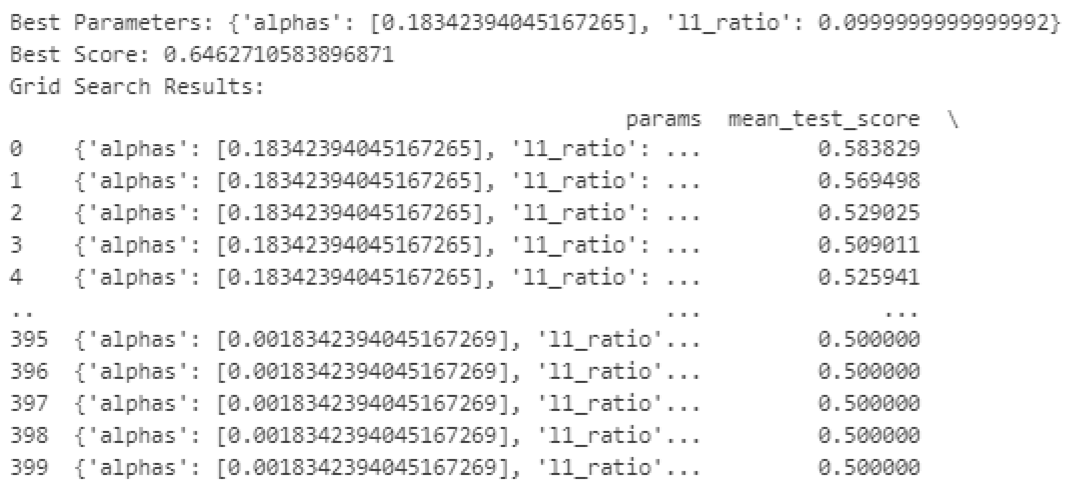
```
274 Index([0.183, 0.11, 0.066, 0.04, 0.024, 0.014, 0.009, 0.005, 0.003, 0.002], dtype='float64')
```



Implementation of grid search with cross-validation to find the best alpha and l1_ratio.

```python
#Steps in elastic net parameter
l1_ratios = np.arange(1, -1, -0.05)
#number of cross fold validations (testing will be 20% and will repeat 5 times to cover all the samples)
cv = KFold(n_splits=5, shuffle=True, random_state=0)
gcv = GridSearchCV((CoxnetSurvivalAnalysis(alpha_min_ratio=1)),
    param_grid={"alphas": [[v] for v in alphas], "l1_ratio" : l1_ratios},
    cv=cv,
    error_score=0.5,
    n_jobs=-1, #to work on the GPU
).fit(X, Y)
cv_results = pd.DataFrame(gcv.cv_results_)

# Display the relevant information
print("Best Parameters:", gcv.best_params_)
print("Best Score:", gcv.best_score_)
print("Grid Search Results:")
print(cv_results[['params', 'mean_test_score', 'std_test_score']])
```

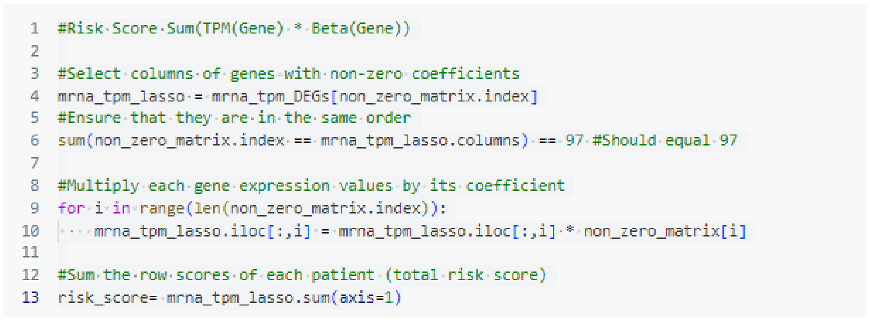
- Prints best parameters and scores.



```
Best Parameters: {'alphas': [0.18342394045167265], 'l1_ratio': 0.0999999999999992}
Best Score: 0.6462710583896871
Grid Search Results:
                                               params  mean_test_score  \
0    {'alphas': [0.18342394045167265], 'l1_ratio': ...         0.583829
1    {'alphas': [0.18342394045167265], 'l1_ratio': ...         0.569498
2    {'alphas': [0.18342394045167265], 'l1_ratio': ...         0.529025
3    {'alphas': [0.18342394045167265], 'l1_ratio': ...         0.509011
4    {'alphas': [0.18342394045167265], 'l1_ratio': ...         0.525941
..                                                 ...              ...
395  {'alphas': [0.0018342394045167269], 'l1_ratio'...         0.500000
396  {'alphas': [0.0018342394045167269], 'l1_ratio'...         0.500000
397  {'alphas': [0.0018342394045167269], 'l1_ratio'...         0.500000
398  {'alphas': [0.0018342394045167269], 'l1_ratio'...         0.500000
399  {'alphas': [0.0018342394045167269], 'l1_ratio'...         0.500000
```

3. Risk Score Calculation:

    - Selects genes with non-zero coefficients.
    - Calculates risk scores by multiplying gene expression with coefficients and summing.

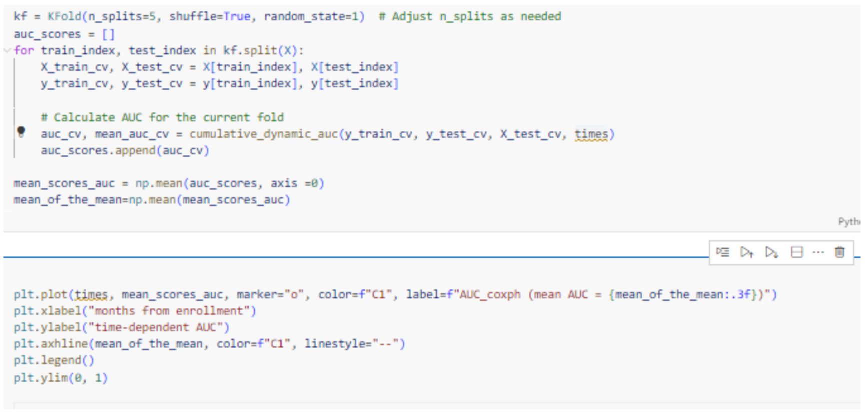


```python
#Risk Score Sum(TPM(Gene) * Beta(Gene))

#Select columns of genes with non-zero coefficients
mrna_tpm_lasso = mrna_tpm_DEGs[non_zero_matrix.index]
#Ensure that they are in the same order
sum(non_zero_matrix.index == mrna_tpm_lasso.columns) == 97 #Should equal 97

#Multiply each gene expression values by its coefficient
for i in range(len(non_zero_matrix.index)):
    mrna_tpm_lasso.iloc[:,i] = mrna_tpm_lasso.iloc[:,i] * non_zero_matrix[i]

#Sum the row scores of each patient (total risk score)
risk_score= mrna_tpm_lasso.sum(axis=1)
```

    - Gene signature is the column of coefficients produced from the lasso-cox function and what is calculated for each patient is the hazard based on his/her signature

4. Time-Dependent AUC Evaluation:

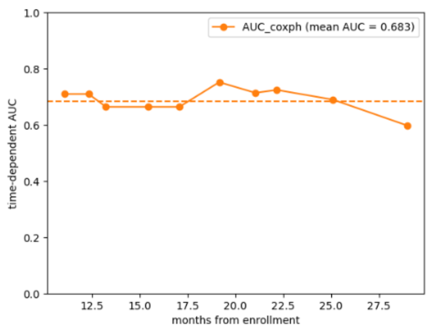
    - Splits data into training and testing sets (5Kfold).



```python
kf = KFold(n_splits=5, shuffle=True, random_state=1)  # Adjust n_splits as needed
auc_scores = []
for train_index, test_index in kf.split(X):
    X_train_cv, X_test_cv = X[train_index], X[test_index]
    y_train_cv, y_test_cv = y[train_index], y[test_index]

    # Calculate AUC for the current fold
    auc_cv, mean_auc_cv = cumulative_dynamic_auc(y_train_cv, y_test_cv, X_test_cv, times)
    auc_scores.append(auc_cv)

mean_scores_auc = np.mean(auc_scores, axis =0)
mean_of_the_mean=np.mean(mean_scores_auc)
```

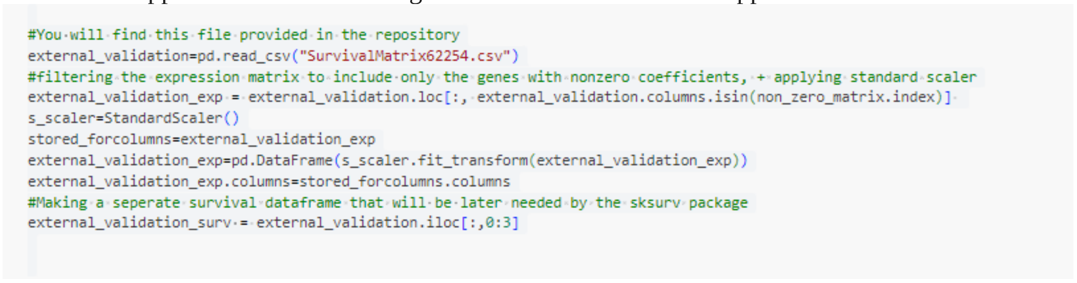
```python
plt.plot(times, mean_scores_auc, marker="o", color=f"C1", label=f"AUC_coxph (mean AUC = {mean_of_the_mean:.3f})")
plt.xlabel("months from enrollment")
plt.ylabel("time-dependent AUC")
plt.axhline(mean_of_the_mean, color=f"C1", linestyle="--")
plt.legend()
plt.ylim(0, 1)
```

    - Calculates and plots time-dependent AUC for risk scores.



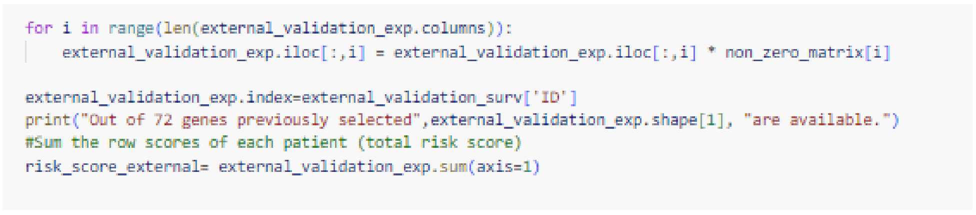
5. External Validation:

   - The GSE62254 file contained both, the expression and survival data in separate columns. Filters were applied to select relevant genes and normalization was applied



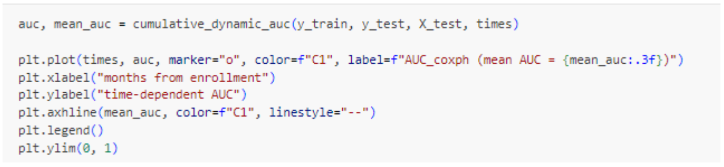
```
#You will find this file provided in the repository
external_validation=pd.read_csv("SurvivalMatrix62254.csv")
#filtering the expression matrix to include only the genes with nonzero coefficients, + applying standard scaler
external_validation_exp = external_validation.loc[:, external_validation.columns.isin(non_zero_matrix.index)]
s_scaler=StandardScaler()
stored_forcolumns=external_validation_exp
external_validation_exp=pd.DataFrame(s_scaler.fit_transform(external_validation_exp))
external_validation_exp.columns=stored_forcolumns.columns
#Making a seperate survival dataframe that will be later needed by the sksurv package
external_validation_surv = external_validation.iloc[:,0:3]
```

   - Calculates risk scores for external data.

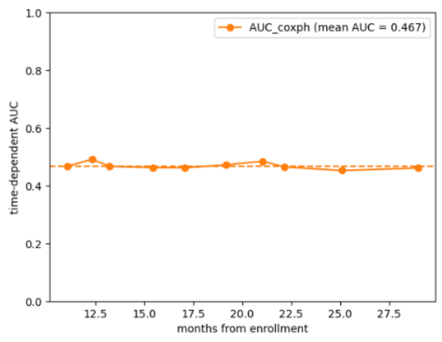


```
for i in range(len(external_validation_exp.columns)):
    external_validation_exp.iloc[:,i] = external_validation_exp.iloc[:,i] * non_zero_matrix[i]

external_validation_exp.index=external_validation_surv['ID']
print("Out of 72 genes previously selected",external_validation_exp.shape[1], "are available.")
#Sum the row scores of each patient (total risk score)
risk_score_external= external_validation_exp.sum(axis=1)
```

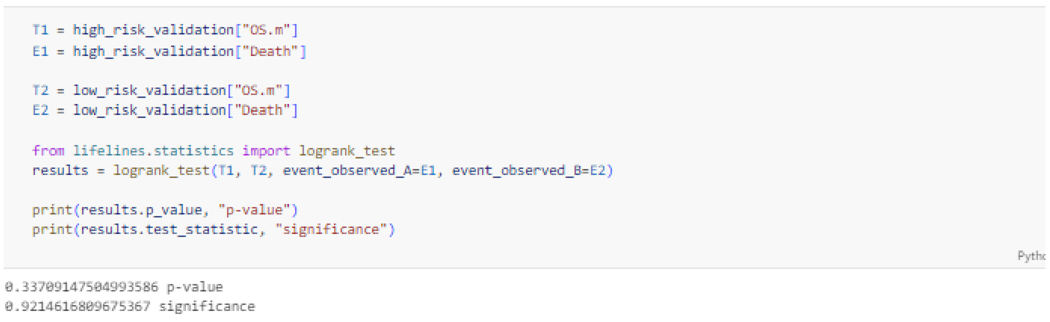
   - Evaluate using time-dependent AUC on external data.



```
auc, mean_auc = cumulative_dynamic_auc(y_train, y_test, X_test, times)

plt.plot(times, auc, marker="o", color=f"C1", label=f"AUC_coxph (mean AUC = {mean_auc:.3f})")
plt.xlabel("months from enrollment")
plt.ylabel("time-dependent AUC")
plt.axhline(mean_auc, color=f"C1", linestyle="--")
plt.legend()
plt.ylim(0, 1)
```

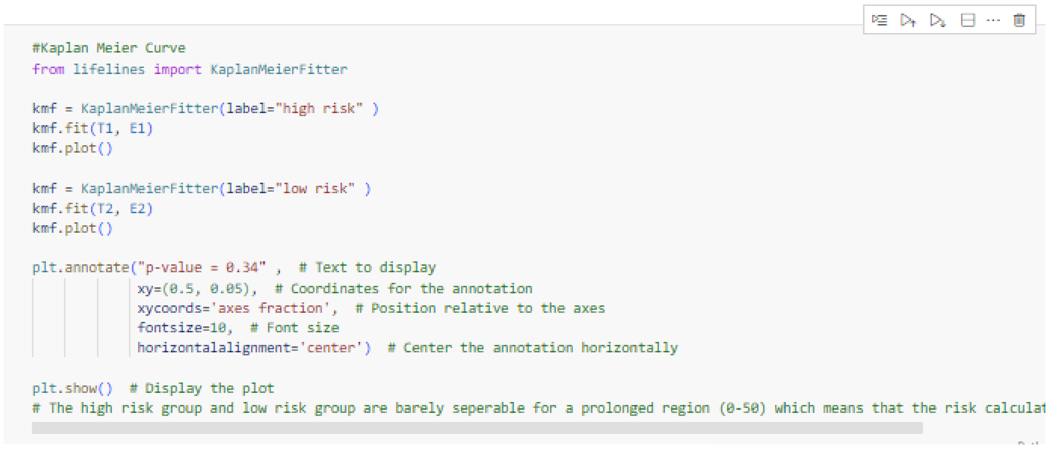


   - Evaluate using the Log Rank Test

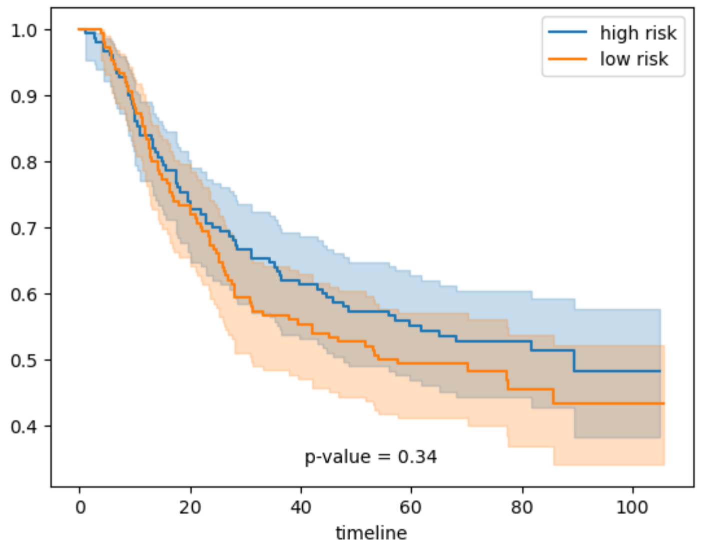


```
T1 = high_risk_validation["OS.m"]
E1 = high_risk_validation["Death"]

T2 = low_risk_validation["OS.m"]
E2 = low_risk_validation["Death"]

from lifelines.statistics import logrank_test
results = logrank_test(T1, T2, event_observed_A=E1, event_observed_B=E2)

print(results.p_value, "p-value")
print(results.test_statistic, "significance")
```

```
0.33709147504993586 p-value
0.9214616809675367 significance
```



```
#Kaplan Meier Curve
from lifelines import KaplanMeierFitter

kmf = KaplanMeierFitter(label="high risk" )
kmf.fit(T1, E1)
kmf.plot()

kmf = KaplanMeierFitter(label="low risk" )
kmf.fit(T2, E2)
kmf.plot()

plt.annotate("p-value = 0.34" ,  # Text to display
             xy=(0.5, 0.05),  # Coordinates for the annotation
             xycoords='axes fraction',  # Position relative to the axes
             fontsize=10,  # Font size
             horizontalalignment='center')  # Center the annotation horizontally

plt.show()  # Display the plot
# The high risk group and low risk group are barely seperable for a prolonged region (0-50) which means that the risk calculat
```



-