# Part 1: Nowcasting

*In this assignment you should nowcast investment using the variables found in file Nowc and in Table 1.*

*1.Nowcast investment using an extending window where the first training set is from 2004Q1 - 2014Q4 . Then nowcast from 2015Q1 to the end using an extending window.*

*2. Nowcast investment using a relaxed LASSO approach where the variables selected from the first LASSO model (from 2004Q1 - 2014Q4) is used as predictors and the estimation method is Ordinary least squares.*

*3. Use an autoregressive model with 2 lags of investment to nowcast investment to compare.*

*4. Discuss the results and which method that is best.*

## Q1. Extending window

The analysis includes data standardization, model fitting, and evaluation of prediction accuracy through various error metrics. The investment data was standardized to have a mean of 0 and a standard deviation of 1. This step is crucial for ensuring that the model treats all features equally, especially when they are on different scales.

The nowcasting process involved creating a training matrix and iterating through the validation set to calculate predictions. The process starts off with the dataset partitioned into a training set (the first 39 observations) and a validation set (the remaining 28 observations). This split is intended to allow for the evaluation of the model's performance on unseen data, as the validation set is not used during the model fitting process. The feature names printed indicate the variables that were deemed most relevant for predicting the target variable (Inv) by the Lasso regression model. The coefficient values in model.coef_ show the relative importance of each selected feature in the predictive model. Features with non-zero coefficients are considered to have a significant influence on the target variable, while features with zero coefficients were effectively removed from the model. The alpha value of ( 0.0645) represents the optimal level of regularization, as determined by the AIC criterion. This parameter controls the trade-off between model complexity and goodness of fit, with higher values of alpha leading to a more parsimonious model with fewer selected features. However, in our case because our alpha value was lower it led to a more prolix model with only one variable excluded.

The second part of the assignment involves nowcasting 2015Q1 to the end. We began by first creating a vector of iteration indices to represent the number of observations to be included in the expanding training set, capturing the evolving dynamics of the underlying economic processes. we then initialize three error metric lists (MSE, ME, and MAE) to evaluate the performance of the model at each iteration. We proceeded to create the training and testing datasets, with the training set expanding at each iteration by including an additional observation from the testing set. The LassoLars Information Criterion (LassoLarsIC) model is re-estimated on the expanded training set, and the nowcast is computed using the updated model coefficients. The differences between the actual and predicted values are used to calculate the error metrics, which are stored for further analysis. The final output is the sequence of MSE values. We averaged the sequence to make it interpretable and had an MSE average of 0.4332 across all iterations. This is a pretty good value because generally MSE values closer to zero represent good model performance. The average MSE value also suggests that the model is able to explain a significant portion of the variability in the investment.

**Nowcasting code**

```
# Standardize the data
sdf= (Nowcast-Nowcast.mean())/Nowcast.std()

# Lets partition our data
    ## The training set equals
sdftrain=sdf[0:39]
    ##The validation set equals
Sdftest=sdf[40:68]

# Find tuning parameter via info criteria
    ## Pick the independent variables and dependent ones
x= sdftrain[Nowcast.columns.difference(['Inv'])]
y = sdftrain['Inv']
    ## Estimate the model
model = LassoLarsIC(criterion='aic', fit_intercept=False)
model.fit(x,y)
print('\n', x. columns)
print('\n', model.coef_)
print('\n', model.alpha_)
 # Nowcasting extending window
    ##Create a vector of iterations
iteration = np.linspace(0,len(sdftest)-1,len(sdftest))
iteration1=iteration.astype(int)

    ## Create help matrix for training set
```

```
MSEt=[]
MEt=[]
MAEt=[]

    ##Create training matrix
xtrain=sdf[sdf.columns.difference(['Inv'])]
ytrain = sdf['Inv']

    ## Create training matrix:
xt=sdftest[sdf.columns.difference(['Inv'])]
yt = sdftest['Inv']

## Write a loop with different values
for i in iteration1:
    xtrain1= xtrain.iloc[0:(39+i)]
    ytrain1= ytrain.iloc[0:(39+i)]
    modeltrain = LassoLarsIC(criterion='aic', fit_intercept=False)
    modeltrain.fit(xtrain1,ytrain1)
    coeftrain=modeltrain.coef_
    xtp=xt.iloc[i]
    ytp=yt.iloc[i]
    nowc=sum(xtp*coeftrain)
    me1=ytp-nowc
    mae1=abs(ytp-nowc)
    mse1=(ytp-nowc)**2
    MEt.append(me1)
    MAEt.append(mae1)
    MSEt.append(mse1)
    print(mse1)

    avg_mse = sum(MSEt) / len(MSEt)

    # Print the average MSE
print('\n',f"Average MSE: {avg_mse:.4f}")
```

```
Average MSE: 0.4332
```

## Q2. Relaxed Lasso

We began our analysis with the standardization of the data, a common preprocessing step that ensures all features are on a similar scale (James et al., 2013). This is particularly important for machine learning algorithms, as it can improve their performance by mitigating the effects of different feature magnitudes (Hastie, Tibshirani, & Friedman, 2009). The standardized dataset is then used to construct the training and testing sets, with the training set expanding at each iteration to capture the evolving dynamics of the underlying economic processes (Braga-Neto, 2020).

The core of the algorithm is the implementation of two regression models: a linear regression model and a LASSO (Least Absolute Shrinkage and Selection Operator) regression model. The linear regression model serves as a baseline, providing a benchmark for comparison (Montgomery, Peck, & Vining, 2012). The LASSO model, on the other hand, is a more sophisticated technique that combines feature selection and regularization, making it well-suited for handling multicollinearity and identifying the most important predictors (Tibshirani, 2011).

Aligned with Limberg et al. (2020) we refined the LASSO model by performing a 10-fold cross-validation to determine the optimal value of the regularization parameter (alpha). This step is crucial, as it ensures that the model strikes a balance between complexity and predictive performance, thereby enhancing its generalization capabilities (Arlot & Celisse, 2010). The results of the cross-validated LASSO model was then used to fit a final, optimized LASSO model, which is again subjected to 10-fold cross-validation to evaluate its overall performance. The cross-validation produced a more parsimonious model as it shrunk the number of significant variables to just six.

Finally, we implemented a "relaxed LASSO" approach, where a linear regression model was fitted using only the six selected features from the previous step with non-zero coefficients from the optimized LASSO model. The relaxed Lasso model provides several important metrics that can be used to evaluate the quality and performance of the model. These metrics include the Akaike Information Criterion (AIC), the Bayesian Information Criterion (BIC), and the coefficient of determination (R-squared). As observed our model has AIC of 90.7 and BIC of 94.03. Generally, a lower AIC and BIC value indicates a better model suggests that the model has a good balance between fit and complexity as well as explanatory power of the labels on investment as a dependent variable. The R-squared value of 0.642 indicates that the relaxed Lasso model is able to explain approximately 64.2% of the variation in the investment variable. This R-squared value further suggests a moderately strong relationship between the selected predictors and the investment variable.

**Relaxed Lasso code**

```
# Standardize the data
sdf= (Nowcast-Nowcast.mean())/Nowcast.std()

# Pick predictors:
```

```
x2 =sdftrain[sdf.columns.difference(['Inv'])]
    # Step 2: Pick output variable:
y2 = sdftrain['Inv']

## Fit the model:
lm = LinearRegression()
model = lm.fit(x2, y2)

# 10 fold cross validation on linear model
scores = cross_val_score(model, x2, y2,
cv=10,scoring='neg_mean_squared_error')
print(scores)
print('\n', scores.mean())

     # LASSO Model
lasso1 = Lasso(alpha=1)
lasso1.fit(x2,y2)
# Show coefficients:
print('\n', lasso1.coef_)

selected_features = x.columns[lasso1.coef_ != 0]

# Lets cross validate our first lasso model
from sklearn.linear_model import LassoCV
lasso1cv = LassoCV(cv=10, random_state=0,max_iter=10000)
lasso1cv.fit(x2, y2)

print('\n', lasso1cv.alpha_) # this value is the optimal alpha
print('\n', lasso1cv.coef_) # This prints 10 coefficients of our cross validated model

# Fit original model (lasso1) with optimal aplha
lasso1_optimized = Lasso(alpha=0.0645)
lasso1_optimized.fit(x2,y2)

# We perform a K-10 cross validaion on lasso1_optimized model
lasso1_optimizedscores = cross_val_score(lasso1_optimized, x2, y2,
cv=10,scoring='neg_mean_squared_error')
print(lasso1_optimizedscores) # returns 10 values represent the performance of the model on
each validation fold
print(lasso1_optimizedscores.mean()) # Returns the overall performance of our model accross
the 10 fold

    # Relaxed Lasso model
# Define our new x-variables
x3 = sdftrain[selected_features]
y2 = sdftrain['Inv']

import statsmodels.api as sm
x3 = sm.add_constant(x3) # creats a new variable y3 which includes the constant

# Run post Lasso model
relaxed_lasso = sm.OLS(y2, x3)
relaxed_lassoresults = post_lasso.fit()
```

```
print(post_lassoresults.summary())


Alpha value: 0.0645211986413993

   OLS Regression Results
==============================================================================
Dep. Variable:                    Inv   R-squared:                       0.642
Model:                            OLS   Adj. R-squared:                  0.632
Method:                 Least Squares   F-statistic:                     66.28
Date:                Wed, 09 Oct 2024   Prob (F-statistic):           9.08e-10
Time:                        12:52:51   Log-Likelihood:                -43.350
No. Observations:                  39   AIC:                             90.70
Df Residuals:                      37   BIC:                             94.03
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0026      0.121      0.022      0.983      -0.243       0.248
InvL1          0.8012      0.098      8.141      0.000       0.602       1.001
==============================================================================
Omnibus:                        3.515   Durbin-Watson:                   1.643
Prob(Omnibus):                  0.172   Jarque-Bera (JB):                2.565
Skew:                          -0.618   Prob(JB):                        0.277
Kurtosis:                       3.227   Cond. No.                         1.25
==============================================================================
```

## Q3. Auto regressive model with 2 lags

We implimented an autoregressive (AR) model with 2 lags to analyze and forecast investment data in the near-term. We standardized data like we did in the other models to ensure the variables are on a similar scale. The AR(2) model was then fitted to the standardized dataset using the AutoReg class from the statsmodels.tsa.ar_model module. The fitted model was then used to generate predictions for the same range as the original data, and the mean squared error (MSE) error metrics was calculated to assess the model's performance.

Additionally, the Akaike Information Criterion (AIC) and Bayesian Information Criterion (BIC) are reported to allow us to compare the models performance with earlier models that we created. The results, including the MSE, are stored in a DataFrame and displayed. The autoregressive model produced AIC and BIC values of 140.20, and 149.02 respectively with MSE of 0.421146. We observe that the MSE is closer to zero suggesting that the model is quite good at predicting investments. However, we compare the MSE, AIC and BIC in the next section with those of other models that we developed to make an informed decission.

### Autoregressive model code

```
# Standardize the data
sdf = (Nowcast - Nowcast.mean()) / Nowcast.std()

# Prepare the data for autoregressive model
investment_data = sdf['Inv']

# Fit the autoregressive model with 2 lags
model_ar = AutoReg(investment_data, lags=2).fit()

# Make predictions for the same range as the original data
predictions_ar = model_ar.predict(start=2, end=len(investment_data)-1, dynamic=False)

# Calculate error metrics
actual_values = investment_data[2:]
ME_ar = np.mean(actual_values - predictions_ar)
MAE_ar = np.mean(np.abs(actual_values - predictions_ar))
MSE_ar = np.mean((actual_values - predictions_ar) ** 2)

# Create a DataFrame for the results
results_ar = pd.DataFrame({ 'MSE': [MSE_ar]})

# Display the results
print(f"AIC: {model_ar.aic:.2f}")
print(f"BIC: {model_ar.bic:.2f}")
print(results_ar)
```

```
AIC: 140.20
BIC: 149.02
        MSE
0  0.421146
```

## Q4. Choosing the best Model

Using the AIC and BIC as the benchmarks for model evaluation. As can be observed both the AIC and BIC values of the relaxed LASSO model are lower than the values for the autoregressive model. Hence, the relaxed LASSO model is a better predictor of investment than the other models. When evaluating the AIC and BIC values, it's important to consider the context of the data being modeled. In the case of the investment nowcasting model presented in the code, the AIC value of 90.70 and the BIC value of 94.03 may initially seem relatively large, and one could question whether they represent a good fit to the data.

However, we recognize that the AIC and BIC values are not inherently good or bad on their own. They are relative measures used to compare the performance of different models. Compared to the autoregressive model the relaxed lasso has a lower AIC and BIC as against the autoregressive model 140.20 and 149.02 respectively. The extending window model is also a bad predictor of investment because as we can observe the average MSE across all predictions is 0.4332 which is slightly above that of the autoregressive model MSE of 0 0.4211. Relaxed Lasso is the model of choice followed by autoregressive model lastly the extending window

## Part 2: Splines

*In this section we are going to use the file waged.csv. In the file age and wage and the logs of the two variables can be found.*

1. *Fit a polynomial regression with degree 1 - 10 where age is independent variable and wage dependent variable. Which degree of polynomial is best?*
2. *Fit a linear, cubic spline and natural spline where age is independent variable and wage dependent variable. Try a different number of knots.*
3. *Which model is best? Comment on the result*

### Q1. Polynomial Regression: Degrees 1 to 10

To determine the optimal model for predicting wage based on age, we began by fitting polynomial regressions of varying degrees and subsequently explored spline-based models. This analysis aimed to assess the performance of different regression techniques, identifying the one that minimizes prediction error and best explains the relationship between the variables.

We started by fitting polynomial regressions, where age was the independent variable and wage was the dependent variable. Using polynomial expansions of degree 1 to 10, we aimed to identify which degree provides the best predictive performance.

A linear regression model was fit to the data for each polynomial degree, and 10-fold cross-validation was employed to calculate the Mean Squared Error (MSE) for each degree. The results clearly compared the models' performances.

**Polynomial regression code**

```
df = pd.read_csv(r'C:\Users\pontu\Downloads\wage.csv', sep=';')
predictors= ['age']
z = df[predictors]
y = df['wage']
poly = PolynomialFeatures(2)
zsq= poly.fit_transform(z)
lm = LinearRegression(fit_intercept=False)
modelsq = lm.fit(zsq,y)
scoressq = cross_val_score(modelsq, zsq,
y,cv=10,scoring='neg_mean_squared_error')
print(scoressq)
print('\n','\n','Mean Score', abs(scoressq.mean()),'\n','\n',)
degree = np.linspace(1,10,10)
degree1=degree.astype(int)
MSEs=[]
lm = LinearRegression(fit_intercept=False)
for d in degree1:
    polyd = PolynomialFeatures(d)
    zd= polyd.fit_transform(z)
    modeld = lm.fit(zd,y)
    scoresd = cross_val_score(modeld, zd, y,cv=10,scoring='neg_mean_squared_error')
    sc= scoresd.mean()
    MSEs.append(sc)
print('Shows Mean Square Error of Degress 1 - 10:','\n', MSEs,'\n')
```

```
Output:
Shows Mean Square Error of Degrees 1 - 10:


 [-784.776507445001, -735.441298385675, -734.9433004181568, -735.127056515429,
-735.2481581548334, -735.7357476135562, -736.745829596839, -735.7898485846974,
-735.8886581147489, -746.4317199833583]
```

Examining the mean square error of each degree shows that degree 3 has the lowest mean square error, 734.943. This demonstrates that a third-degree polynomial provides the best fit for predicting wage based on age, balancing model complexity and predictive accuracy. Higher-degree polynomials did not improve the model's performance, potentially overfitting the data.

## Splines: Linear, Cubic, and Natural

We explored the performance of three spline regression models—linear spline, cubic spline, and natural spline—to assess their comparison to polynomial regression. Each spline model was evaluated using 10-fold cross-validation to determine its predictive accuracy.

When determining knots for the linear and cubic spline models, the numbers 25, 40, and 60 were chosen as a baseline. Each value was then fine-tuned by changing it up or down by one until the models no longer performed better by changing any value.

### Q1. Linear Spline

A piecewise linear regression was applied with knots at ages 26, 40, and 58. This approach fits a linear model between each knot, allowing for changes in slope at the specified knot points.

```
transformed_z1 = dmatrix("bs(z, knots=(26, 40, 58), degree=1, include_intercept=False)",
{"zt1": z}, return_type='dataframe')

lm1 = LinearRegression(fit_intercept=False)
model1 = lm1.fit(transformed_z1, y)

scores1 = cross_val_score(model1, transformed_z1, y, cv=10,
scoring='neg_mean_squared_error')
print('Mean Absolute Score (Linear spline):', abs(scores1.mean()))
```

```
Mean Absolute Score (Linear Spline): 773.601
```

### Q2. Cubic Spline

The cubic spline model was built with knots at ages 19, 30, and 66. Unlike the linear spline, this model fits cubic polynomials between each knot, allowing for more flexibility and smoother transitions.

```
transformed_z2 = dmatrix("bs(z, knots=(19, 30, 66), degree=3, include_intercept=False)",
{"zt2": z}, return_type='dataframe')

model2 = lm1.fit(transformed_z2, y)

scores2 = cross_val_score(model2, transformed_z2, y, cv=10,
scoring='neg_mean_squared_error')
print('Mean Absolute Score (Cubic spline):', abs(scores2.mean()))
```

```
Mean Absolute Score (Cubic Spline): 673.909
```

## Q3. Natural Spline

We also applied a natural spline model with 6 degrees of freedom. Natural splines are cubic with additional boundary constraints to avoid overfitting at the edges.

```
transformed_z3 = dmatrix("cr(z, df=6, include_intercept=False)", {"zt3": z},
return_type='dataframe')

model3 = lm1.fit(transformed_z3, y)
scores3 = cross_val_score(model3, transformed_z3, y, cv=10,
scoring='neg_mean_squared_error')
print('Mean Absolute Score (Natural spline):', abs(scores3.mean()))

Mean Absolute Score (Natural Spline): 661.984
```

## Q4. Dummy Variable Model

As a baseline comparison, we used a dummy variable model that divides the age data into four quantiles. Each quantile was treated as a separate category. While this model captures distinct jumps between groups, it lacks the smooth transitions and flexibility offered by spline models.

```
cut_age = pd.qcut(df['age'], 4)
Dummy = pd.get_dummies(cut_age)
modeldummy = lm1.fit(Dummy, y)
# Cross-validation for dummy variable model
scoress = cross_val_score(modeldummy, Dummy, y, cv=10, scoring='neg_mean_squared_error')
print('Mean Absolute Score (Dummy variable model):', abs(scoress.mean()))


Mean Absolute Score (Dummy Variable Model): 772.0445693964514
```

## Conclusion

The natural spline with 6 degrees of freedom achieved the lowest Mean Absolute Score (661.984), suggesting it best balances model complexity and predictive accuracy. The cubic spline followed closely, with a Mean Absolute Score of 673.909. The linear spline was less flexible, with a Mean Absolute Score of 773.601, which makes it perform worse than our dummy variable at 772.045.

This indicates that while the linear spline captures some non-linearity in the data, it may still oversimplify the relationship between age and wage, especially when compared to the cubic and natural splines. The dummy variable model, despite being simpler, performed better than the linear spline, showing that dividing the age variable into discrete groups might capture the wage pattern more effectively than forcing a piecewise linear relationship.

## References

Arlot, S., & Celisse, A. (2010). A survey of cross-validation procedures for model selection. *Statistics Surveys*, 4, 40–79.

Braga-Neto, U. (2020). *Fundamentals of Pattern Recognition and Machine Learning* (1st ed. 2020.). Springer Nature. https://doi.org/10.1007/978-3-030-27656-0

Friedman, J., Hastie, T., & Tibshirani, R. (2010). Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1), 1–22.

Hastie, T., Tibshirani, R., & Friedman, J. (2011). *The elements of statistical learning* (2nd ed.). Springer.

Limberg, C., Wersing, H., & Ritter, H. (2020). Beyond Cross-Validation—Accuracy Estimation for Incremental and Active Learning Models. *Machine Learning and Knowledge Extraction*, *2*(3), 327–346. https://doi.org/10.3390/make2030018

Montgomery, D. C., Peck, E. A., & Vining, G. G. (2012). *Introduction to linear regression analysis* (5th ed.). Wiley.