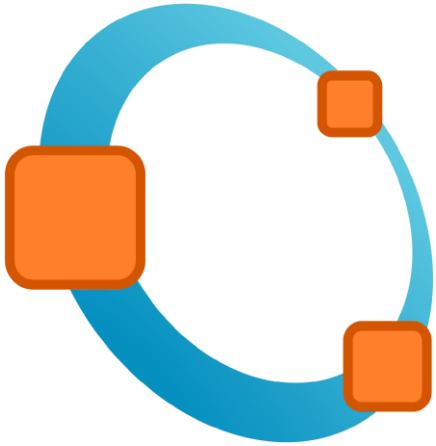
Google Summer of Code 2026

PROPOSAL



GNU OCTAVE

LinearModel and CompactLinearModel CLASS OBJECTS

PERSONAL DETAILS:

Name: Avanish Salunke	**Telephone:**
Nickname: Avi	**Email:** avanishsalunke16@gmail.com
GitHub: AvanishSalunke	**Time Zone:** IST (GMT+5:30)
Discourse: Avanish	**Language:** English
LinkedIn: Avanish Salunke	**Country:** India

INTRODUCTION:

GNU Octave is a high-level language that is mainly used for **numerical computations**. It is a command-line interface for numerically solving **linear** and **nonlinear** problems, as well as performing experiments using a language that is mostly compatible with MATLAB. Octave is easily extensible and customizable through user-written functions or dynamically loadable **modules** written in C++, C, and Fortran. As free software under the **GNU GPL**, Octave thrives on community **contributions** to expand its usefulness.

Linear regression is a fundamental statistical technique used to establish the relationship between a dependent **response variable** (*y*) and one or more independent **predictor variables** (*X*). The linear regression equation is given by:

$$\mathbf{y} = \mathbf{X\beta} + \boldsymbol{\varepsilon}, \quad \boldsymbol{\varepsilon} \sim \mathbf{N}(\mathbf{0}, \boldsymbol{\sigma}^2\mathbf{I})$$

In this equation, ***X*** is the design matrix (our data), $\boldsymbol{\beta}$ is the unknown coefficients we wish to estimate, and $\boldsymbol{\varepsilon}$ is the normally distributed random error term.

The goal of the model is to calculate the coefficients $\boldsymbol{\beta}$ required to minimize the error using the Ordinary Least Squares (OLS) approach. The mathematical formula for the above approach is $\boldsymbol{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$, but as the above formula involves the calculation of the **inverse of a matrix**, it is highly **unstable**. The widely accepted approach is the **QR decomposition** of the matrix X=QR. This approach provides a highly stable solution:

$$\boldsymbol{\beta} = \boldsymbol{R}^{-1}\boldsymbol{Q}^T\boldsymbol{y}$$

Once the **coefficients** are estimated, the model's accuracy must be evaluated using standard statistical metrics:

- ➔ ***Sum of Squares***: We calculate the residual error (SSE), the variance explained by the model (SSR), and the total variance (SST).
- ➔ ***R^2 and adjusted R^2***: These metrics use the sums of squares to indicate the proportion of variance in the data that the model successfully explains.
- ➔ ***Covariance Matrix***: We compute the covariance of the estimates, $\text{Cov}(\boldsymbol{\beta}) = \mathbf{MSE} \cdot (\mathbf{R}^T\mathbf{R})^{-1}$ This is strictly required to calculate confidence intervals and perform hypothesis testing (p-values) on the coefficients.

1) The LinearModel is used during model **development**. It is a heavy object that stores the complete dataset, residuals, and deep diagnostics needed to validate and refine the math.
2) The CompactLinearModel is used for fast, memory-efficient predictions. Created via the compact () method, it strips away the heavy training data and keeps only the essential mathematical coefficients and covariance.

PROJECT DESCRIPTION: (ISSUE)

Linear regression is a widely used statistical tool. While MATLAB provides a modern object-oriented interface for it, Octave's current fitlm function just wraps anovan and returns basic cells and structs. This project will close that gap by bringing a proper object-oriented regression interface to Octave.

The main goals of the project are:

I. *LinearModel Class*: Create the LinearModel classdef. This will act as the **full regression model** containing the training data, residuals, diagnostics, and **methods** to manipulate the model.

II. *CompactLinearModel Class*: Create the CompactLinearModel classdef. This will be a lightweight, prediction-only version of the model that drops the heavy training data to save **memory**.

III. *Rewriting fitlm*: Update the existing **fitlm.m** function so it returns a proper **LinearModel** object.

IV. *Supporting Modern Inputs*: Extend the fitlm function so it can accept table data types and Wilkinson formula strings (like 'y ~ x1 + x2').

V. *Implementing stepwiselm*: Create a new stepwiselm.m function for stepwise regression. It will use criterion-based selection (like **AIC** or **BIC**) to find the best model and return a LinearModel object.

EXAMPLE EXPECTED OUTPUT:

```
octave:20> x1 = [1; 2; 3; 4; 5; 6];
octave:21> x2 = [5; 1; 4; 2; 6; 3];
octave:22> x3 = [2; 5; 1; 6; 3; 4];
octave:23> y  = [12; 15; 14; 22; 25; 20];
octave:24> X = [x1, x2];
octave:25> mdl_matrix = fitlm(X, y)
mdl_matrix =

Linear regression model:
    y ~ 1 + x1 + x2

Estimated Coefficients:
                   Estimate       SE       tStat       pValue
                   ________    ______    _______    ________

    (Intercept)      9.5263     4.532      2.102      0.12632
    x1               2.2105    0.9011     2.4531     0.091423
    x2              0.21053    0.9011    0.23363      0.83031

Number of observations: 6, Error degrees of freedom: 3
Root Mean Squared Error: 3.76
R-squared: 0.674,  Adjusted R-Squared: 0.457
F-statistic vs. constant model: 3.11, p-value = 0.186

octave:26> tbl = table(x1, x2, y, 'VariableNames', {'x1', 'x2', 'y'});
octave:27> mdl_formula = fitlm(tbl, 'y ~ x1 + x2')
mdl_formula =

Linear regression model:
```

```
    y ~ 1 + x1 + x2

Estimated Coefficients:
                    Estimate       SE       tStat      pValue
                    ________    _______    _______    _________

    (Intercept)      9.5263      4.532      2.102      0.12632
    x1               2.2105     0.9011     2.4531     0.091423
    x2              0.21053     0.9011    0.23363      0.83031

Number of observations: 6, Error degrees of freedom: 3
Root Mean Squared Error: 3.76
R-squared: 0.674,  Adjusted R-Squared: 0.457
F-statistic vs. constant model: 3.11, p-value = 0.186

octave:28> mdl_formula.Rsquared
ans =

  scalar structure containing the fields:

    Ordinary = 0.6745
    Adjusted = 0.4575

octave:29> cMdl = compact(mdl_formula)
cMdl =

Compact linear regression model:
    y ~ 1 + x1 + x2

Estimated Coefficients:
                    Estimate       SE       tStat      pValue
                    ________    _______    _______    _________

    (Intercept)      9.5263      4.532      2.102      0.12632
    x1               2.2105     0.9011     2.4531     0.091423
    x2              0.21053     0.9011    0.23363      0.83031

Number of observations: 6, Error degrees of freedom: 3
Root Mean Squared Error: 3.76
R-squared: 0.674,  Adjusted R-Squared: 0.457
F-statistic vs. constant model: 3.11, p-value = 0.186

octave:30> Xnew = table([7; 8], [4; 5], 'VariableNames', {'x1', 'x2'});
octave:31> yhat = predict(cMdl, Xnew)
yhat =

   25.842
   28.263

octave:32> tbl_step = table(x1, x2, x3, y, 'VariableNames', {'x1', 'x2', 'x3',
'y'});
octave:33> mdlStep = stepwiselm(tbl_step, 'y ~ 1', 'Upper', 'linear', 'Verbose', 0)
mdlStep =

Linear regression model:
    y ~ 1 + x1

Estimated Coefficients:
                    Estimate       SE       tStat      pValue
                    ________    _______    _______    _________

    (Intercept)        10.2      3.0554     3.3384     0.028879
    x1               2.2286     0.78454     2.8406      0.04684
```

```
Number of observations: 6, Error degrees of freedom: 4
Root Mean Squared Error: 3.28
R-squared: 0.669,  Adjusted R-Squared: 0.586
F-statistic vs. constant model: 8.07, p-value = 0.0468
```

CURRENT IMPLEMENTATION:

A. ***fitlm.m***: Wraps anovan entirely. Returns `[T, STATS]` - a cell array and struct, not a model object. No table, formula, or robust fitting support.

B. ***stepwisefit.m***: Forward-backward selection by **p-value**. Returns raw vectors, not a model object. No **AIC/BIC** criterion support.

C. ***anovan.m***: The actual fitting engine behind fitlm. Contains an internal lmfit function doing QR-based weighted least squares. Returns a **STATS** struct with coefficients, residuals, design matrix, and Cook's distance.

D. ***parseWilkinsonFormula.m***: This function, which I implemented, is a recursive descent parser for **Wilkinson notation** and can be used to build **design matrices** from tables. However, there is a currently existing **bug** related to the sorting of the variables. The function currently alphabetizes the variables and performs an ascending sortrows on the binary matrix instead of maintaining the column order as the table was originally entered, as MATLAB does. This bug must be fixed so the coefficient ordering matches MATLAB's output exactly.

```
octave:5> schema = parseWilkinsonFormula('Y ~ x1 * x2 * x3', 'matrix');
octave:6> disp(schema.Terms)
   0   0   0   0
   0   0   0   1
   0   0   1   0
   0   1   0   0
   0   0   1   1
   0   1   0   1
   0   1   1   0
   0   1   1   1
```

PROPOSED SOLUTION:

The core of this solution involves building two **distinct classes** to separate the heavy model training phase from the lightweight prediction phase.

The implementation will be structured as follows:

A. *CLASS ARCHITECTURE*:

I. **LinearModel**: LinearModel is the heavy, training-centric model. It will be produced by fitlm or stepwiselm through `mdl = LinearModel.fit (X, y);`

II. **CompactLinearModel**: CompactLinearModel is the lightweight, prediction-only model. It will be created by calling `cMdl = compact (mdl);`

III. **Memory Optimization**: The CompactLinearModel constructor will copy only the essential structural components: the **coefficient vector**, **covariance matrix**, **R factor**, **formula**, and **variable metadata**. The massive training data, residuals, diagnostics, and the Q factor are intentionally stripped away to save memory.

B. *REWRITING FITLM FUNCTION*:

All parsing and fitting logic will live inside `LinearModel.fit()`, which will handle **six** specific input signatures:

1) `mdl = fitlm (X, y)` - will fit a basic linear model using a **predictor matrix** X and a **response vector** y.

```
X = [1 2; 2 4; 3 5];
y = [1; 2; 3];
mdl = fitlm(X, y)
```

2) `mdl = fitlm (X, y, modelspec)` - will fit a model using a **matrix** and **vector**, while allowing us to specify the exact model type (like 'linear' or 'interactions').

```
X = [1 2; 2 4; 3 5];
y = [1; 2; 3];
mdl = fitlm(X, y, 'interactions')
```

3) `mdl = fitlm (tbl)` - will fit a model directly from a **table** where the last column is automatically treated as the **response variable**.

```
x1 = [1; 2; 3]; x2 = [2; 4; 5]; y = [1; 2; 3];
tbl = table(x1, x2, y);
mdl = fitlm(tbl)
```

4) `mdl = fitlm (tbl, respname)` - will fit a model from a **table** while explicitly defining which column name is the **response variable**.

```
x1 = [1; 2; 3]; x2 = [2; 4; 5]; y = [1; 2; 3];
tbl = table(x1, x2, y);
mdl = fitlm(tbl, 'y')
```

5) `mdl = fitlm (tbl, formula)` - will fit a model from a **table** using a Wilkinson formula string to define the exact relationship between the variables.

```
x1 = [1; 2; 3]; x2 = [2; 4; 5]; y = [1; 2; 3];
tbl = table(x1, x2, y);
mdl = fitlm(tbl, 'y ~ x1 + x2')
```

6) `mdl = fitlm (tbl, y vec)` - will fit a model using a **table** containing only the **predictor variables**, paired with a separate **response vector**.

```
x1 = [1; 2; 3]; x2 = [2; 4; 5]; y = [1; 2; 3];
tbl_predictors = table(x1, x2);
mdl = fitlm(tbl_predictors, y)
```

It also fully supports the following Name-Value pairs: `CategoricalVars, Exclude, Intercept, PredictorVars, ResponseVar, RobustOpts, VarNames, and Weights`.

C. *IMPLEMENTING STEPWISELM FUNCTION*:

- ➔ The **stepwiselm.m** function will implement automated stepwise regression and will return a complete LinearModel object. It will accept all the standard fitlm input signatures, plus several powerful new **Name-Value pairs** used specifically for automated model selection: `PEnter`, `PRemove`, `Upper`, `Lower`, `Criterion` (supporting 'sse', 'aic', 'bic', 'rsquared', 'adjrsquared'), `NSteps`, and `Verbose`.
- ➔ At each step, the **algorithm** will evaluate all candidate term **additions** or **removals**, it will apply the single best change if it meets the specified criterion threshold, and will repeat this process until **convergence**.
- ➔ **Sample Use Cases**:
  1) Basic Stepwise Selection -

```
octave:196> x1 = [1; 2; 3; 4]; x2 = [4; 3; 2; 1]; y = [2; 4; 6; 8];
octave:197> tbl = table (x1, x2, y);
octave:198> mdl = stepwiselm (tbl)
```

  2) Specifying Model Bounds and Criterion -

```
octave:199> x1 = [1; 2; 3; 4]; x2 = [4; 3; 2; 1]; y = [2; 4; 6; 8];
octave:200> tbl = table (x1, x2, y);
octave:201> mdl = stepwiselm (tbl, 'y ~ 1', 'Upper', 'linear',
'Criterion', 'bic')
```

  3) Controlling Thresholds and Output Verbosity -

```
octave:202> x1 = [1; 2; 3; 4]; x2 = [4; 3; 2; 1]; y = [2; 4; 6; 8];
octave:203> tbl = table (x1, x2, y);
octave:204> mdl = stepwiselm (tbl, 'PEnter', 0.05, 'Verbose', 0)
```

D. *CLASS PROPERTIES*:

I. CompactLinearModel:

```
Coefficients, CoefficientCovariance, CoefficientNames, NumCoefficients,
NumEstimatedCoefficients, DFE, MSE, RMSE, SSE, SSR, SST, Rsquared,
ModelCriterion, LogLikelihood, ModelFitVsNullModel, Formula,
```

```
NumPredictors, NumVariables, NumObservations, PredictorNames,
ResponseName, VariableNames, VariableInfo, Robust.
Hidden: Coefs, R_qr, Qy, Rtol, Terms, CoefTerm, HasIntercept,
DesignMeans.
```

II. **LinearModel**:

```
All compact properties (duplicated), plus: Fitted, Residuals,
Diagnostics, Variables, ObservationInfo, ObservationNames, Leverage,
Steps.
Hidden: y_r, design_r, w_r, Q_qr, Design.
```

E. *CLASS METHODS*:

I. **CompactLinearModel**:

- ➔ **predict (X_{new}, NV) -** It will return predicted values and optional confidence/prediction intervals.
- ➔ **feval (x_1, x_2, ….) -** Evaluate the model as a function of individual predictor vectors.
- ➔ **coefCI (alpha) -** Calculate coefficient confidence intervals: Estimate ± tinv (1-α/2, DFE) × SE.
- ➔ **coefTest (H, c) -** Perform an F-test on the hypothesis H*b = c.
- ➔ **anova(type) -** Generate an ANOVA table.
- ➔ **random (Xnew) -** Generate predicted values plus normal noise scaled by RMSE.
- ➔ **compact () -** It will return itself (the compact object).
- ➔ **disp () -** Prints the formatted model summary.

II. **LinearModel**: (**Inherited:** All methods from CompactLinearModel and the following)

- ➔ **compact () -** Strip heavy data and returns a new CompactLinearModel(this) object.
- ➔ **addTerms (terms) -** Refit the model with added terms and returns a new LinearModel.
- ➔ **removeTerms (terms) -** It will refit the model with removed terms and returns a new LinearModel.
- ➔ **step (NV) -** Perform a single stepwise selection from current model.
- ➔ **plotResiduals (plottype) -** Generate residual plots (histogram, fitted, probability, lagged).
- ➔ **plotDiagnostics (plottype) -** It will generate diagnostic plots.
- ➔ **plotAdded (var) -** Generate an added variable plot.
- ➔ **dwtest (option, tail) -** Perform the Durbin-Watson test for autocorrelation.

F. ***CLASSDEF STRUCTURE***: (LinearModel and CompactLinearModel)

```
classdef LinearModel
    ## Full Linear Regression Model Object

    properties (SetAccess = private)
        ## the same are shared with CompactLinearModel classdef
        Coefficients              ## Table: Estimate, SE, tStat, pValue with RowNames.
        CoefficientCovariance     ## p-by-p covariance matrix of coefficient estimates.
        CoefficientNames          ## Cell array of coefficient names.
        NumCoefficients           ## Total number of coefficients in the model.
        NumEstimatedCoefficients ## Non-redundant coefficients (rank of design matrix).
        DFE                       ## Degrees of freedom for error.
        MSE                       ## Mean squared error: SSE / DFE.
        RMSE                      ## Root mean squared error: sqrt(MSE).
        SSE                       ## Error (residual) sum of squares.
        SSR                       ## Regression sum of squares.
        SST                       ## Total sum of squares: SSR + SSE.
        Rsquared                  ## Struct with fields Ordinary and Adjusted.
        ModelCriterion            ## Struct with fields AIC, AICc, BIC, CAIC.
        LogLikelihood             ## Log-likelihood of the fitted model.
        ModelFitVsNullModel       ## Struct with Fstat, Pvalue, NullModel.
        Formula                   ## Wilkinson notation formula string.
        NumPredictors             ## Number of predictor variables.
        NumVariables              ## Total variable count: predictors + response.
        NumObservations           ## Observations used in the fit.
        PredictorNames            ## Cell array of predictor variable names.
        ResponseName              ## Name of the response variable.
        VariableNames             ## Cell array of all variable names.
        VariableInfo              ## Struct describing each variable's type.
        Robust                    ## Struct with RobustWgtFun, Tune, Weights; or empty.

        ## this are unique to LinearModel classdef
        Fitted                    ## n-by-1 vector of fitted values.
        Residuals                 ## Struct: Raw, Pearson, Standardized, Studentized.
        Diagnostics               ## Struct: Leverage, CooksDistance, Dffits, CovRatio.
        Variables                 ## Training data stored as struct or table.
        ObservationInfo           ## Struct: Weights (n-by-1) and Subset (logical).
        ObservationNames          ## Cell array of row names, or empty.
        Leverage                  ## Diagonal of Hat matrix, clamped to [0, 1].
        Steps                     ## Stepwise history struct, or empty.
    endproperties

    properties (SetAccess = private, Hidden)
        Coefs                     ## Raw coefficient vector (p x 1).
        R_qr                      ## R factor from QR decomposition.
        Qy                        ## Q' * y product from QR.
        Rtol                      ## Rank tolerance used during fitting.
        Terms                     ## Binary terms matrix.
        CoefTerm                  ## Coefficient-to-term index map.
        HasIntercept              ## True if model includes an intercept.
        DesignMeans               ## Column means of design matrix.
        y_r                       ## Response vector for valid rows.
        design_r                  ## Design matrix for valid rows.
        w_r                       ## Weight vector for valid rows.
        Q_qr                      ## Q factor from QR (for diagnostics).
        Design                    ## Full design matrix.
    endproperties

    methods
        function obj = LinearModel (varargin)
            ## Constructor: called internally by LinearModel.fit.
            ## Signature 1: fit(X, y)               - matrix + response vector
            ## Signature 2: fit(X, y, modelspec)    - with model specification
            ## Signature 3: fit(tbl)                - table, last column = response
            ## Signature 4: fit(tbl, name)          - table + response column name
            ## Signature 5: fit(tbl, formula)       - table + Wilkinson formula
            ## Signature 6: fit(tbl, y_vec)         - table + separate response
            ## All signatures accept trailing Name-Value pairs.
        endfunction
```

```
        function disp (obj)
            ## Prints formatted model summary to the command window.
        endfunction

        function [ypred, yci] = predict (obj, Xnew, varargin)
            ## Predicted values and optional confidence/prediction intervals.
        endfunction

        function ypred = feval (obj, varargin)
            ## Evaluates model from individual predictor vectors.
        endfunction

        function ci = coefCI (obj, alpha)
            ## Coefficient confidence intervals: Estimate +/- tinv * SE.
        endfunction

        function [p, F, r] = coefTest (obj, H, c)
            ## Delegates to standalone coefTest function.
        endfunction

        function ysim = random (obj, Xnew)
            ## Predicted values plus normal noise scaled by RMSE.
        endfunction

        function cMdl = compact (obj)
            ## Creates a CompactLinearModel, stripping data and diagnostics.
        endfunction

        function newMdl = addTerms (obj, terms)
            ## Refits with added terms; returns a new LinearModel.
        endfunction

        function newMdl = removeTerms (obj, terms)
            ## Refits with removed terms; returns a new LinearModel.
        endfunction

        function newMdl = step (obj, varargin)
            ## Stepwise term selection from the current model.
        endfunction

        function tbl = anova (obj, sstype)
            ## Integrates with the anova class once available.
        endfunction

        function [p, stat] = dwtest (obj, option, tail)
            ## Delegates to standalone dwtest function.
        endfunction

        function h = plotResiduals (obj, plottype, varargin)
            ## Residual plots: histogram, fitted, probability, lagged.
        endfunction

        function h = plotDiagnostics (obj, plottype, varargin)
            ## Diagnostic plots: leverage, cookd, contour.
        endfunction

        function h = plotAdded (obj, var, varargin)
            ## Added variable (partial regression) plot.
        endfunction
    endmethods

    methods (Static, Hidden)
        function mdl = fit (X, varargin)
            ## Static factory: parses inputs, builds design matrix,
            ## fits via lsfit, computes all stats, returns LinearModel.
        endfunction
    endmethods

endclassdef
```

```
classdef CompactLinearModel
    ## Compact Linear Regression Model Object

    properties (SetAccess = private)
        Coefficients              ## Table: Estimate, SE, tStat, pValue with RowNames.
        CoefficientCovariance     ## p-by-p covariance matrix of coefficient estimates.
        CoefficientNames          ## Cell array of coefficient names.
        NumCoefficients           ## Total number of coefficients in the model.
        NumEstimatedCoefficients ## Non-redundant coefficients (rank of design matrix).
        DFE                       ## Degrees of freedom for error.
        MSE                       ## Mean squared error: SSE / DFE.
        RMSE                      ## Root mean squared error: sqrt(MSE).
        SSE                       ## Error (residual) sum of squares.
        SSR                       ## Regression sum of squares.
        SST                       ## Total sum of squares: SSR + SSE.
        Rsquared                  ## Struct with fields Ordinary and Adjusted.
        ModelCriterion            ## Struct with fields AIC, AICc, BIC, CAIC.
        LogLikelihood             ## Log-likelihood of the fitted model.
        ModelFitVsNullModel       ## Struct with Fstat, Pvalue, NullModel.
        Formula                   ## Wilkinson notation formula string.
        NumPredictors             ## Number of predictor variables.
        NumVariables              ## Total variable count: predictors + response.
        NumObservations           ## Observations used in the fit.
        PredictorNames            ## Cell array of predictor variable names.
        ResponseName              ## Name of the response variable.
        VariableNames             ## Cell array of all variable names.
        VariableInfo              ## Struct describing each variable's type.
        Robust                    ## Struct with RobustWgtFun, Tune, Weights; or empty.
    endproperties

    properties (SetAccess = private, Hidden)
        Coefs                     ## Raw coefficient vector (p x 1).
        R_qr                      ## R factor from QR decomposition.
        Qy                        ## Q' * y product from QR.
        Rtol                      ## Rank tolerance used during fitting.
        Terms                     ## Binary terms matrix.
        CoefTerm                  ## Coefficient-to-term index map.
        HasIntercept              ## True if model includes an intercept.
        DesignMeans               ## Column means of design matrix.
    endproperties

    methods
        function obj = CompactLinearModel (Mdl)
            ## Constructor: copies prediction-essential properties from a
            ## LinearModel object. Returns blank object if called empty.
            ## Validates that input is a LinearModel; errors otherwise.
        endfunction

        function disp (obj)
            ## Prints compact model summary to the command window.
        endfunction

        function [ypred, yci] = predict (obj, Xnew, varargin)
            ## Predicted values and optional confidence/prediction intervals.
        endfunction

        function ypred = feval (obj, varargin)
            ## Evaluates model from individual predictor vectors.
        endfunction

        function ci = coefCI (obj, alpha)
            ## Coefficient confidence intervals: Estimate +/- tinv * SE.
        endfunction

        function [p, F, r] = coefTest (obj, H, c)
            ## Delegates to standalone coefTest function.
        endfunction
```

```
        function ysim = random (obj, Xnew)
            ## Predicted values plus normal noise scaled by RMSE.
        endfunction

        function cobj = compact (obj)
            ## Returns self (already compact).
        endfunction
    endmethods

endclassdef
```

POSSIBLE DIFFICULTIES AND SOLUTIONS:

1. ***Implementing Robust Fitting***: Since robustfit is not available in Octave, I will have to implement the loop for **robust fitting** and the **weight functions** completely from scratch. This involves dealing with specific mathematical cases where the program may divide by zero or where the variance may be zero, so that the program does not crash.
2. ***Standalone functions***: Functions like coefTest, dwtest can be used as either standalone commands or object methods. To avoid writing the exact same code twice, I will place the main logic in **standalone .m files**, and my class methods will simply act as **wrappers** that call those files.
3. ***Speeding Up Stepwise Regression***: Recalculating the full matrix math from scratch every time stepwiselm tests adding or removing a variable is computationally very slow. To fix this, I will use a mathematical shortcut (QR updating and downdating) to simply adjust the existing calculation, which will save a significant amount of computation time.

BENEFITS TO OCTAVE:

- ➔ Adds LinearModel and CompactLinearModel, both of which are compatible with MATLAB, for the first time in **Octave**.
- ➔ compact () allows for memory-efficient deployment of models by removing **heavy training data**.
- ➔ parseWilkinsonFormula is integrated with regression, allowing for easy fitting of models by simply calling fitlm on a table.
- ➔ stepwiselm allows for automatic selection of models based on a given **criterion**, similar to **AIC** and **BIC**, and returns a proper **model**.
- ➔ Diagnostic plots, for example, **plotResiduals** and **plotDiagnostics**, would be built-in, giving users a quick and easy way to **diagnose** their models.
- ➔ Makes it easier for users to predict by allowing them to simply pass in new data to the **predict** method.

TIMELINE: (REFERENCE)

1. ***CURRENT FOCUS AND PREPARATION***: Until the results are announced, I will be deep diving into the class structures for **Anova** (Project 1), **LinearModel** and **CompactLinearModel** (Project 2). I will be mastering the internal **methods** for these classes. Along with this, I will be actively **contributing** to the statistics package with bug fixes.
2. ***PROJECT PREPARATION***: Since both project issues have been open since **October 19, 2025**, I have invested equal research into both. I have been studying their **internal logic** and working on their dependencies to ensure a smooth implementation for whichever project is selected.
3. ***COMMUNITY BONDING AND EXAMS***: My **end-semester** examinations take place from **May 1 to May 20**. During this period, I will remain available for project discussions and will continue my contributions on **alternate days** between papers to ensure I stay on track.
4. ***POST EXAM FOCUS***: After my exams end on **May 20**, I have a **two-month** summer break with no other commitments. I plan to treat this as a **full-time role**, dedicating **10-12 hours a day** to ensure the project stays ahead of schedule.
5. ***ACADEMIC TERM SCHEDULE***: Once my college session resumes, I will follow a proper schedule to dedicate **2 to 4 hours** every **weekday** and **8 to 12 hours** every **weekend** until the project is completed successfully.

Pre- community bonding period	➔ Implementation of some standalone functions. ➔ Study Octave's **classdef** by examining **existing** classes. ➔ Draft initial class skeletons for **both** classes. ➔ Continue contributing **bug** fixes and resolving dependencies in the **statistics** package.
Community bonding period.	➔ Discuss implementation details with mentor Andreas Bertsatos. ➔ Finalize **property** list and **method** signatures for all methods. ➔ Confirm integration approach. ➔ Stay updated with the latest version of **octave** and corresponding **packages.**
Week 1	**CompactLinearModel Class:** ➔ Set up class with all properties and constructor. ➔ Implement the **predict** method. ➔ Implementing **feval**, **random**, and **disp** methods. ➔ Write initial **BISTs**.
Week 2	**LinearModel Class:** ➔ Set up class with all **properties** (unique + shared). ➔ Implementing the internal **lsfit** function. ➔ Implementing **LinearModel.fit ()** for matrix input (signatures 1 and 2). ➔ Computing SSE, SSR, SST, R^2, F-stat, MSE, RMSE, log-likelihood, AIC/BIC.

Week 3	**Coefficient inference, diagnostics, and compact:** ➔ Implement **coefCI**. ➔ Implement standalone coefTest function. ➔ Implementing the compact () method on LinearModel. ➔ Computing **leverage**, **Cook's distance**, and all four residual types. ➔ **BISTs** for all methods.
Week 4	**Table and formula input support:** ➔ Adding table input support to **LinearModel.fit ()** (signatures 3-6). ➔ Integrating **parseWilkinsonFormula** in **model_matrix** mode. ➔ Handling CategoricalVars, VarNames, PredictorVars, ResponseVar NV pairs. ➔ Writing **BISTs** for all six input signatures.
Week 5	**Rewriting fitlm and formatting disp:** ➔ Rewriting fitlm.m calling **LinearModel.fit ()**. ➔ Formatting **disp ()** output. ➔ Handling **Exclude** and Weights **name-value** pairs. ➔ Rewriting existing fitlm **BISTs** for the new object output.
Week 6	**Buffer:** ➔ Fixing **bugs** found during Weeks 1-5. ➔ Running end-to-end tests. ➔ Tightening **BIST** coverage across all code written so far. ➔ Addressing **mentor** feedback and changes if needed,
Week 7	**Robust fitting via IRLS:** ➔ Implementing the **IRLS** loop with all 8 weight functions. ➔ Handling RobustOpts NV pair. ➔ Handle edge cases and write **BISTs**.
Week 8	**Visualization methods:** ➔ Implementing **plotResiduals**. ➔ Implementing **plotDiagnostics**. ➔ Implementing **plotAdded**. ➔ Verification against MATLAB's output.
Week 9	**stepwiselm implementation:** ➔ Implementing **stepwiselm.m** ➔ Implementing **addTerms**, **removeTerms**, and **step** methods on LinearModel. ➔ Supporting **criterion**-based selection: SSE, AIC, BIC. ➔ Adding **Upper/Lower** model bounds and **PEnter/PRemove** thresholds.
Week 10	**Anova integration, dwtest, and edge cases:** ➔ Implementing standalone **dwtest** function. ➔ Integrating the **anova method** once the **parallel** anova project is ready. ➔ Handling stepwise edge cases.

Week 11	**Integration testing and hardening**: ➔ End-to-end testing. ➔ Expanding **BISTs** coverage. ➔ Documentation. ➔ Fix bugs if any.
Week 12	**Documentation and final submission**: ➔ Expanding **BIST** coverage across all files. ➔ Documentation. ➔ Code cleanup. ➔ Prepare to submit
End of Gsoc	**Coding period ends**! ➔ Edge-cases + fixes (if any) ➔ Code submission ➔ Final review from my potential mentor Andreas Bertsatos.

MY CONTRIBUTIONS IN OCTAVE:

1) **Wilkinson's** formula parser: (PR_1) (ISSUE) (PR_2) **+1688 -0**
2) **Summary method** in CVPartition: (ISSUE) (PR) **+618 -9**
3) **Makima** Interpolation: (ISSUE) (PR_1)(PR_2) **+494 -0**
4) **LIBSVM** 3.25-3.36 upgrade: (ISSUE) (PR_1) (PR_2) **+347 -66**
5) Handling of missing group labels in **anova1**: (PR) **+19 -7**
6) Support for string, categorical and multi-grouping variables in **boxplot**: (PR) **+141 -82**
7) Support another argument in **cmdscale**: (PR) **+148 -46**
8) String, Categorical array, handling of missing values in **confusionmat** (ISSUE) (PR) **+304 -61**
9) Check for 2D matrix in **grp2idx**: (PR) **+16 -1**
10) Fix errors in **randsample**: (PR) **+62 -19**
11) Handle Nan values in **signtest** function: (PR) **+18 -3**
12) Handle Nan values in **signrank** function: (PR) **+10 -6**
13) Check for symmetric and psd matrix in **pcacov**: (PR) **+20 -2**
14) Appropriate errors in **datasample** function: (ISSUE) (PR) **+15 -0**
15) Degrees of freedom issue in **wishrnd** function: (ISSUE) (PR) **+40 -27**
16) Restructuring some functions: (PR_1) (PR_2) (PR_3) **+281 -100**

BACKGROUND AND MOTIVATION:

My name is Avanish Salunke, and I am currently pursuing a **B. tech** in **Information Technology** at **VJTI**, Mumbai. I have been working with **C++** for around five years, along with about a year of experience in **Python**. My background in **data structures** and **competitive programming** has helped me develop a structured approach to problem-solving and writing efficient code.

I started contributing to **GNU Octave** in October, and it has been my first experience working with a large and actively maintained **C++ codebase**. Before this, most of my work in C++ was limited to academic assignments and theoretical challenges. Through Octave, I began understanding how **different modules interact**, how **numerical** and **statistical methods** are implemented in practice. It gave me a much clearer picture of how real systems are built and maintained.

Along with submitting **contributions**, I have also participated in discussions on Octave Discourse, especially in the help category, where I tried to assist **maintainers** with ongoing **code testing**. Over time, I have become familiar with the **statistics package**, its internal structure, and some of its **dependencies**. Working through **review** feedback has also improved my understanding of coding standards and long-term maintainability. I would especially like to thank Andreas for his guidance and detailed reviews.

My commitment to the community is also reflected in my activity on the Octave Discourse; I recently earned the Aficionado **badge** for visiting the forum for over 100 consecutive days. While my current contributions have mainly focused on the statistics package, I am interested in exploring deeper parts of Octave's core architecture in the future. Through **GSOC**, I aim to contribute in a more focused and structured manner, and continue contributing to the Octave even beyond the program.

Thank you for considering my proposal.

- Avanish Salunke

REFERENCES:

1) Fit Linear Regression MATLAB Documentation:
2) MATLAB - what is a linear regression model?
3) MATLAB's N-Way ANOVA (anovan) Documentation:
4) GNU Octave N-Way ANOVA (anovan) Documentation:
5) Linear Regression Model MATLAB Documentation:
6) Compact Linear Regression Model MATLAB Documentation:
7) Compact MATLAB Documentation:
8) Wilkinson's Notation MATLAB Documentation:

9) Stepwise Regression MATLAB Documentation:

10) MATLAB's grp2idx Documentation:

11) Durbin-Watson test MATLAB documentation:

12) QR updating and downdating:

13) CoefTest MATLAB Documentation: