In [1]:

```
from IPython.display import display_html
def display_side_by_side(*args):
    html_str=''
    for df in args:
        html_str+=df.to_html()
    display_html(html_str.replace('table','table style="display:inline"'),raw=True)

class display(object):
    """Display HTML representation of multiple objects"""
    template = """<div style="float: left; padding: 10px;">
    <p style='font-family:"Courier New", Courier, monospace'>{0}</p>{1}
    </div>"""
    def __init__(self, *args):
        self.args = args

    def _repr_html_(self):
        return '\n'.join(self.template.format(a, eval(a)._repr_html_())
                         for a in self.args)

    def __repr__(self):
        return '\n\n'.join(a + '\n' + repr(eval(a))
                           for a in self.args)
```

In [2]:

```
class Master:

    def predectors_select(data_frame, Exception=None):
        predictors = []
        data_type = ['int64', 'float64']
        for column in data_frame.columns:
            if data_frame[column].dtype in data_type:
                if Exception == None:
                    predictors.append(column)
                elif column not in Exception:
                    predictors.append(column)
        return set(predictors)

#
    def correlation(data_frame: 'dataframe', min:'decimal'=.5, max:'decimal'=1):
        data_frame_temp = data_frame.corr().abs().reset_index()
        data_frame_temp.drop(columns=['index'], inplace=True)
        unique_v1 = set()
        unique_v2 = set()
        counter = 0
        for index, data in data_frame_temp.iterrows():
            el=[_ for _ in range(data_frame_temp.index.size) if _!= counter]
            for j in el:
                if data[j] > min and data[j] < max:
                    unique_v1.add(data.index[index])
                    unique_v2.add(data.index[j])
                    #print("{0:>15} {1:>15} {2:>10.5f}".format(data.index[index], data.index[j], data[j]))
            counter +=1

        return sorted(list(unique_v1.intersection(unique_v2))) #unique_v1, unique_v2
#
    def vif(dataframe):
        """
        variance inflation factor (VIF)
        Parameters
        ----------
        data_frame = dataframe

        Returns
        -------
        Pandas Series formed by VIF for each value in the dataframe
        """
        #vif_dataframe = pd.DataFrame()
        #vif_dataframe['VIF Factor'] = [variance_inflation_factor(dataframe.values, i) for i in range(dataframe.shape[1]
        #vif_dataframe['Features']   = dataframe.columns

        vif = dict()
        for idx in range(dataframe.shape[1]):
            vif[ dataframe.columns[idx] ] = variance_inflation_factor(dataframe.values, idx)
        return pd.Series(vif, index=dataframe.columns)
#

    def log_trans(data_frame, to_log=None):
        #data_frame_log = pd.DataFrame()
        if to_log == None:
            data_frame = data_frame.applymap(lambda x: np.log(x) if x > 0 else 0)
        else:
            for feat in to_log:
                data_frame[feat] = data_frame[feat].apply(lambda x: np.log(x))

        #return data_frame
#
    def sklearn_model(x,y):
        x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=.3, random_state=7)

        linreg = LinearRegression()
        linreg.fit(x_train, y_train)

        y_hat_train = linreg.predict(x_train)
        y_hat_test  = linreg.predict(x_test)

        train_residuals = y_hat_train - y_train
        test_residuals = y_hat_test - y_test
```

```
        train_mse = mean_squared_error(y_train, y_hat_train)
        test_mse  = mean_squared_error(y_test, y_hat_test)

        print('R^2 Score {0:>22}'.format(round(linreg.score(x, y),2)))
        print('Train Mean Squarred Error {0:>20}'.format(train_mse))
        print('Test Mean Squarred Error {0:>21}'.format(test_mse))

        return linreg

#
    def metrics(y, y_hat):

        r2   = r2_score(y, y_hat)
        mae  = mean_absolute_error(y, y_hat)
        mse  = mean_squared_error(y, y_hat)
        rmse = np.sqrt(mean_squared_error(y, y_hat))

        y_original     = y_scaler.inverse_transform(y)
        y_hat_original = y_scaler.inverse_transform(np.array(y_hat).reshape(-1,1))

        mae_inverse  = mean_absolute_error( y_original, y_hat_original)
        rmse_inverse = np.sqrt(mean_squared_error( y_original, y_hat_original))

        print(f"{'R-squared':<25}{r2:>5.3f}")
        print(f"{'Mean Absolute Error':<25}{mae:>5.3f} ${mae_inverse:,.2f}")
        print(f"{'Mean Squared Error':<25}{mse:>5.3f}")
        print(f"{'Root Mean Squared Error':<25}{rmse:>5.3f} ${rmse_inverse:,.2f}")

        return mae, mse, rmse

#
    def stepwise_selection(X, y,
                           initial_list=[],
                           threshold_in=0.01,
                           threshold_out = 0.05,
                           verbose=True):
        """ Perform a forward-backward feature selection
        based on p-value from statsmodels.api.OLS
        Arguments:
            X - pandas.DataFrame with candidate features
            y - list-like with the target
            initial_list - list of features to start with (column names of X)
            threshold_in - include a feature if its p-value < threshold_in
            threshold_out - exclude a feature if its p-value > threshold_out
            verbose - whether to print the sequence of inclusions and exclusions
        Returns: list of selected features
        Always set threshold_in < threshold_out to avoid infinite looping.
        See https://en.wikipedia.org/wiki/Stepwise_regression for the details
        """
        included = list(initial_list)
        while True:
            changed=False
            # forward step
            excluded = list(set(X.columns)-set(included))
            new_pval = pd.Series(index=excluded, dtype='float64')
            for new_column in excluded:
                model = sm.OLS(y, sm.add_constant(pd.DataFrame(X[included+[new_column]]))).fit()
                new_pval[new_column] = model.pvalues[new_column]
            best_pval = new_pval.min()
            if best_pval < threshold_in:
                best_feature = new_pval.idxmin()
                included.append(best_feature)
                changed=True
                if verbose:
                    print('Add  {:30} with p-value {:.6}'.format(best_feature, best_pval))

            # backward step
            model = sm.OLS(y, sm.add_constant(pd.DataFrame(X[included]))).fit()
            # use all coefs except intercept
            pvalues = model.pvalues.iloc[1:]
            worst_pval = pvalues.max() # null if pvalues is empty
            if worst_pval > threshold_out:
                changed=True
                worst_feature = pvalues.idxmax()
                # worst_feature = pvalues.argmax()
                included.remove(worst_feature)
                if verbose:
                    print('Drop {:30} with p-value {:.6}'.format(worst_feature, worst_pval))
            if not changed:
                break
        return included
```