Feature Extraction

Prepare the data to build a build, check for missing values, outliers, treat them, also remove the variabled with low standard deviations.

In [ ]:
```
#Square root transformation
#this transformation uis used to reduce skewness in the data
np.sqrt(y_train)
```

In [ ]:
```
#Reciprocal Transformation
(1/y_train)
```

In [1]:
```
#log transformation
np.log(y_train)
```

In [ ]:
```
#Box-cox tranformation
stats.boxcox(y_train)[0][0:5]
```

In [ ]:
```
#Stepwise Regression
'''Stepwise regression is a process that selects the most important
features (independent variables) in the dataset by removing or adding
a variable at every step in the regression. In this section we study two
approaches to perform stepwise regression:

Forward Selection
Backward Elimination
Recursive Feature Elimination'''

#Foward selection
linereg = LinearRegression()

linreg_forward = sfs(estimator = linreg, k_features = 'best', forward = True
                    verbose = 2, scoring = 'r2')
sfs_forward = linreg_foward.fit(x_train, y_train)

#Backward elimination
linreg_backward = sfs(estimator = linreg, k_features = 'best', forward = False
                    verbose = 2, scoring = 'r2')
sfs_backward = linreg_backward.fit(x_train, y_train)

#Recursive Feature Elimination
rfe_model = RFE(estimator = linreg, n_features_to_select =12)
rfemodel = rfe_model.fit(x_train, y_train)
feat_indes = pd.Series(data = rfemodel.ranking_, index = x_train.columns)
sig_feat = feat_indes[feat_indes == 1].index

#create a new training model after RFE and build the model for RFE data
new_X_train = X_train[['T1', 'RH_1', 'T2', 'RH_2', 'T3', 'T4', 'RH_4', 'T7', 'T8'
       'T9', 'Windspeed']]
linreg = LinearRegression()
linreg.fit(new_X_train, y_train)
linreg.score(new_X_train, y_train)
```

In [ ]:
```
#scaling the data if target variable is present separately
y = (df_target - df_target.mean()) / df_target.std()
```

In [ ]:
```
#Genelarized function for model metrics

def get_train_rmse(model):
    train_pred = model.predict(x_train)
    mse_train = mean_squared_error(y_train, train_pred)
    rmse_train = round(np.sqrt(mse_train), 4)
    return(rmse_train)

def get_test_rmse(model):
    test_pred = model.predict(x_test)
    mse_test = mean_squared_error(y_test, test_pred)
    rmse_test = round(np.sqrt(mse_test), 4)
    return(mse_test)

def mape(actual, predicted):
    return(np.mean(np.abs((actual - predicted)/actual))*100)

def get_test_mape(model):
    test_pred = model.predict(X_test)
    mape_test = mape(y_test, test_pred)
    return(mape_test)

def get_score(model):
    r_sq = model.score(X_train, y_train)
    n = X_train.shape[0]
    k = X_train.shape[1]
    r_sq_adj = 1 - ((1-r_sq)*(n-1)/(n-k-1))
    return ([r_sq, r_sq_adj])

score_card = pd.DataFrame(columns=['Model_Name', 'Alpha (Wherever Required)', 'l1
                                   'Adj. R-Squared', 'Test_RMSE', 'Test_MAPE'

def update_score_card(algorithm_name, model, alpha = '-', l1_ratio = '-'):
    global score_card
    score_card = score_card.append({'Model_Name': algorithm_name,
                       'Alpha (Wherever Required)': alpha,
                       'l1-ratio': l1_ratio,
                       'Test_MAPE': get_test_mape(model),
                       'Test_RMSE': get_test_rmse(model),
                       'R-Squared': get_score(model)[0],
                       'Adj. R-Squared': get_score(model)[1]}, ignore_index = Tru

def plot_coefficients(model, algorithm_name):
    df_coeff = pd.DataFrame({'Variable': X.columns, 'Coefficient': model.coef_})
    sorted_coeff = df_coeff.sort_values('Coefficient', ascending = False)
    sns.barplot(x = "Coefficient", y = "Variable", data = sorted_coeff)
    plt.xlabel("Coefficients from {}".format(algorithm_name), fontsize = 15)
    plt.ylabel('Features', fontsize = 15)
```

In [ ]:
```
#updating scorecard
update_score_card(algorithm_name = 'Linear Regression', model = mlr_model)
```

In [ ]:
```
#Cross validation
# Remember: Cross validation works on training set not on test set

#K Fold Cross validation
kf = KFold(n_splits = 5) #no of splits u wana divide data into
def get_score(model, x_train_k, x_test_k, y_train_k, y_test_k):
    model.fit(x_train_k, y_train_k)
    return model.score(x_test_k, y_test_k)

X_train, X_test, y_train, y_test = train_test_split(x, y, random_state = 10, test

scores = []

for train_index, test_index in kf.split(X_train):
    X_train_k, X_test_k, y_train_k, y_test_k = X_train.iloc[train_index], X_trair
                                               y_train.iloc[train_index], y_trair

    scores.append(get_score(LinearRegression(), X_train_k, X_test_k, y_train_k, y

print('All scores: ', scores)
print("\nMinimum score obtained: ", round(min(scores), 4))
print("Maximum score obtained: ", round(max(scores), 4))
print("Average score obtained: ", round(np.mean(scores), 4))

#the above KFold cross validation can be performed on cross_val_score funtion

scores = cross_val_score(estimator = LinearRegression(),
                         X = X_train,
                         y = y_train,
                         cv = 5,
                         scoring = 'r2')
```

In [ ]:
```
#Leave one out cross validation - LOOCV

'''It is a process in which the model is trained on the training dataset,
with the exception of only one data point, which is used to test the
model. It is a process in which the model is trained on the training
dataset, with the exception of only one data point, which is used to test
the model.'''


loocv_rmse = []
loocv = LeaveOneOut()

# use the for loop to build the regression model for each cross validation
# use split() to split the dataset into two subsets; one with (n-1) data points a
# where, n = total number of observations

for train_index, test_index in loocv.split(X_train):
    X_train_l, X_test_l, y_train_l, y_test_l = X_train.iloc[train_index], X_train
                                               y_train.iloc[train_index], y_train
    linreg = LinearRegression()
    linreg.fit(X_train_l, y_train_l)

    mse = mean_squared_error(y_test_l, linreg.predict(X_test_l))
    rmse = np.sqrt(mse)
    loocv_rmse.append(rmse)

print("\nMinimum rmse obtained: ", round(min(loocv_rmse), 4))
print("Maximum rmse obtained: ", round(max(loocv_rmse), 4))
print("Average rmse obtained: ", round(np.mean(loocv_rmse), 4))
```

In [ ]:
```
#Gradient Descent
#Stochastic Gradient Descent
'''The gradient descent method considers all the data points to calculate
the values of the parameters at each step'''

sgd = SGDRegressor(random_state = 10)
linreg_with_SGD = sgd.fit(X_train, y_train)
print('RMSE on train set:', get_train_rmse(linreg_with_SGD))
print('RMSE on test set:', get_test_rmse(linreg_with_SGD))

update_score_card(algorithm_name = 'Linear Regression (using SGD)', model = linre


'''The coefficients obtained from SGD have smaller values as compared to
the coefficients obtained from linear regression using OLS.'''
```

In [ ]:
```
#regularization

'''One way to deal with the overfitting problem is by adding the Regularization t
It is observed that inflation of the coefficients cause overfitting.
Penalties are added'''

#Ridge Regression
# use Ridge() to perform ridge regression
# 'alpha' assigns the regularization strength to the model
# 'max_iter' assigns maximum number of iterations for the model to run
ridge = Ridge(alpha = 1, max_iter = 500)
ridge.fit(X_train, y_train)
get_test_rmse(ridge)
update_score_card(algorithm_name='Ridge Regression (with alpha = 1)', model = ric

#The coefficients obtained from ridge regression have smaller values as
#compared to the coefficients obtained from linear regression using OLS.
```

In [ ]:
```
#Lasso regression
'''Lasso regression shrinks the less important variable's coefficient to zero whi
a type of regularization technique that uses L1 norm for regularization.'''

lasso = Lasso(alpha = 0.01, max_iter = 500)
lasso.fit(X_train, y_train)
get_test_rmse(lasso)

#printing the variables with 0 co-eff after lasso reg
df_lasso_coeff = pd.DataFrame({'Variable': X.columns, 'Coefficient': lasso.coef_}
df_lasso_coeff.Variable[df_lasso_coeff.Coefficient == 0].to_list()
```

In [ ]:
```
#Elastic Net Regression
'''This technique is a combination of Rigde and Lasso reression techniques. It
considers the linear combination of penalties for L1 and L2 regularization.'''

enet = ElasticNet(alpha = 0.1, l1_ratio = 0.01, max_iter = 500)
enet.fit(X_train, y_train)
get_test_rmse(enet)
```

In [ ]:
```
#finding Optimal value for alpha for all resgression in regularization

#ridge regression
tuned_paramaters = [{'alpha':[1e-15, 1e-10, 1e-8, 1e-4,1e-3, 1e-2, 0.1, 1, 5, 10,
ridge = Ridge()
ridge_grid = GridSearchCV(estimator = ridge,
                          param_grid = tuned_paramaters,
                          cv = 10)
ridge_grid.fit(X_train, y_train)
print('Best parameters for Ridge Regression: ', ridge_grid.best_params_, '\n')
print('RMSE on test set:', get_test_rmse(ridge_grid))

#Lasso Regression
lasso = Lasso()
lasso_grid = GridSearchCV(estimator = lasso,
                          param_grid = tuned_paramaters,
                          cv = 10)
lasso_grid.fit(X_train, y_train)

print('Best parameters for Lasso Regression: ', lasso_grid.best_params_, '\n')
print('RMSE on test set:', get_test_rmse(lasso_grid))

#Elastic Net Regression

tuned_paramaters = [{'alpha':[0.0001, 0.001, 0.01, 0.1, 1, 5, 10, 20, 40, 60],
                      'l1_ratio':[0.0001, 0.0002, 0.001, 0.01, 0.1, 0.2]}]
enet = ElasticNet()
enet_grid = GridSearchCV(estimator = enet,
                          param_grid = tuned_paramaters,
                          cv = 10)
enet_grid.fit(X_train, y_train)

print('Best parameters for Elastic Net Regression: ', enet_grid.best_params_, '\r
print('RMSE on test set:', get_test_rmse(enet_grid))

'''finally visulize the score of all the model
1. Ridge Regression using Grid search CV
2. Lasso Regression using Grid search CV
3. Elastic Net Regression using Grid search CV
4. Linear Regression using SGD
5. Lasso Regression
6. Ridge Regression with alpha 2
7. Ridge Regression with alpha 1
8. Linear Regression model

finalize the model that has lowest RMSE as this model best resolves the
overfitting problem.'''
```

In [ ]: