Train and Test Data

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,random_state=0)
```

Data Preprocessing

Standardization

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler().fit(X_train)

standardized_X_train = scaler.transform(X_train)
standardized_X_test = scaler.transform(X_test)
```

Normalization

```
from sklearn.preprocessing import Normalizer
scaler = Normalizer().fit(X_train)

normalized_X = scaler.transform(X_train)
normalized_X_test = scaler.transform(X_test)
```

Imputing Missing Values

```
from sklearn.preprocessing import Imputer

imp = Imputer(missing_values=0, strategy='mean', axis=0)
imp.fit_transform(X_train)
```

Linear Regression

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \ldots + \beta_n x_n + \varepsilon$$

Linear Regression Using Sklearn

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
```

Linear Regression Using statmodels

```
import statsmodels.api as sm

lmodel = sm.OLS(y_train, X_train)
```

Model Fitting

```
model.fit(X_train, y_train)
```

Prediction

```
y_pred = model.predict(X_test)
```

Feature Selection

Forward Selection

```
from mlxtend.feature_selection import SequentialFeatureSelector as sfs
linreg_forward = sfs(estimator = model, k_features, forward = True)
```

Backward Elimination

```
from mlxtend.feature_selection import SequentialFeatureSelector as sfs
linreg_backward = sfs(estimator = model, k_features, forward = False)
```

Recursive Forward Elimination

```
from sklearn.feature_selection import RFE
rfe_model = RFE(estimator = model, n_features_to_select)
```

Cross Validation

Cross_Val_Score

```
from sklearn.model_selection import cross_val_score

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

K-Fold CV

```
from sklearn.model_selection import KFold
kf = KFold(n_splits)
```

Leave One Out CV

```
from sklearn.model_selection import LeaveOneOut

loocv = LeaveOneOut()
```

Hyperparameter Tuning

GridSearchCV

```
from sklearn.model_selection import GridSearchCV
grid_model = GridSearchCV(estimator, param_grid , cv )
```

Gradient Descent

Stochastic Gradient Descent

```
from sklearn.linear_model import SGDRegressor
sgd = SGDRegressor()
```

Regularization

Ridge Regularization

```
from sklearn.linear_model import Ridge
ridge = Ridge(alpha)
```

Lasso Regularization

```
from sklearn.linear_model import Lasso
lasso = Lasso(alpha)
```

Elastic-Net Regularization

```
from sklearn.linear_model import ElasticNet
enet = ElasticNet(alpha, l1_ratio)
```

Model Evaluation

R-Squared

```
from sklearn.metrics import r2_score
r_sq = r2_score(y_true, y_pred)
```

Adjusted R-Squared

```
n = number of observations
k = number of columns (including intercept)

adj_r_squared = 1 - (((1 - r_sq) * (n - 1)) / (n - k - 1))
```

Evaluating Model Performance

Mean Squared Error

```
from sklearn.metrics import mean_squared_error
MSE = mean_squared_error(y_test, y_pred)
```

Root Mean Squared Error

```
RMSE = np.sqrt(MSE)
```

Mean Absolute Error

```
from sklearn.metrics import mean_absolute_error
MAE =  mean_absolute_error(y_true, y_pred)
```

Mean Absolute Percentage Error

```
MAPE = np.mean(np.abs((y_test - y_pred) / y_test)) * 100
```