***ASAD ASHRAF KAREL***

# All syntaxes for Statistics:

## Bionomial, Poission, Normaml Distribution

In [ ]:
```
# Binomail Distribution
st.binom.cdf/sf/pmf(x,n,p)

# Poisson Distribution
st.poisson.cdf/sf/pmf(x,mean)

# Normal Distribution
st.norm.cdf/sf/pmf(x,mean,std)

st.norm.isf(0.7,mean,std) # Exmple: At least 70% customers have balance in t
st.norm.ppf(0.3,mean,std)

```

## Calculation for ZScore value

In [ ]:
```
z1 = mean + std
z2 = mean - std

st.norm.cdf(z1,mean,std) - st.norm.cdf(z2,mean,std)
# 0.68

z1 = mean + 2*std
z2 = mean - 2*std

st.norm.cdf(z1,mean,std) - st.norm.cdf(z2,mean,std)
# 0.95

z1 = mean + 3*std
z2 = mean - 3*std

st.norm.cdf(z1,mean,std) - st.norm.cdf(z2,mean,std)
# 0.99
```

## Sampling Error

### Difference between the Pop Mean and the Sample Mean

In [ ]:
```
np.mean(population) - np.mean(sample)
```

# Confidence Interval with z distribution

In [ ]:
```
# Std Error:
Std_Error = sigma/np.sqrt(n)

# OR:
Std_Error = st.sem(sample) # np.std(mins, ddof = 1)/np.sqrt(n)

```

### Z_values Estimation:

In [ ]:
```
st.norm.isf(0.95)  # for 95% area

# With upper limit:
st.norm.isf(0.95) , st.norm.ppf(0.95) , st.norm.interval(0.95) # Here we get

# Upperlimit - Lower limit: (CONFIDENCE INTERVAL)
values(UL,LL) = Mean - (z_alpha * std) , Mean + (z_alpha * std)

# Direct Estimation of Confidence Interval:
st.norm.interval(0.95, mean)
### OR m - st.norm.isf(0.025) , m + st.norm.isf(0.025) Mathematically
```

# Confidence interval with t distribution

In [ ]:
```
# t_critical_values:
st.t.isf(0.99, df = n-1), st.t.ppf(0.99, df = n-1) # For 99%.

# CI for t distribution:
t_stat = st.t.ppf(0.99, df = n-1)
SE = s/np.sqrt(n)

CI = xbar - (SE*t_stat) , xbar + (SE*t_stat)

# Direct Estimation of Confidnce Interval:
st.t.interval(0.95, dof= n-1, xbar, SE)
```

# Hypothesis Technique

**Complete process:**

**If population std is given:**

In [ ]:
```

# Given data:
Mu =
n =
sigma =
xbar =
alpha =

### Critical Value:
zcrit = stats.norm.isf(0.975).round(3) # 0.025

# Z_stats:
std_error=sigma/np.sqrt(n)

z_stat=(xbar-Mu)/std_error

print(zcrit , z_stat)

## If z_stat > zcrit, we reject our Null hypothesis.

# With P_value calculations:
p_value=st.norm.cdf(z_stat)*2
print(p_value)

## If P_value > alpha, we accept our Null hypothesis.
```

**If proportion is given:**

In [ ]:
```
zcrit = st.norm.isf(0.05)
print(zcrit)

# Proportion:
p1 = x/n
q = 1 - p

zstat = (p1 - p) / np.sqrt(p*q/n)
print(zstat)

if zcrit > zstat :
    print('\nWe are failed to reject null hypothesis.')
    print('Student internship is across 45%.')
else:
    print('\nWe reject our null hypothesis.')
    print('Student internship is about our expectations.')
```

**For T test:**

In [ ]:
```
# Test Statistic
xbar =
popmean =
std_error = sigma/np.sqrt(n)
# OR
std_error = stats.sem(data)

teststats = (xbar-popmean)/ std_error
print(teststats)

# Remember that in Python, the Manual PValue Calc is One Sided. Thus, we nee
1 - stats.t.cdf(teststats, n-1)

# Since P Value > 0.05, then we Fail to Reject the Claim.
```

## OR

In [ ]:
```
# First we go for sample std.dev if std.dev is not given:
sd = np.std(sample, ddof=1)

std_error = sd/np.sqrt(n)
# OR
std_error = stats.sem(data)

t_stat = (xbar-Mean)/std_error

t_crit = st.t.isf(alpha/2,n-1) # If two tail test.

# If t_stat < t_crit, hence data comes in accept region.

P_value = st.t.sf(t_stat,n-1)*2  # 2 tail test
# OR
1 - stats.t.cdf(teststats, n-1)

# If Pvalue > alpha, hence we accept null hypotheis.

# One line code, rather doing all mathematics:
st.ttest_1samp(sample,Mean)
```

**To check the normality of data, whether it is normal or not:**

In [ ]:
```
st.shapiro(data)

# If Pvalue>Alpha: Which means, data follows normal distribution. Hence we a
# Hence we approach non parametric test.
```

In [ ]:
```
# Pvalue>Alpha
# Now test for variance equality:

st.levene(data1,data2)

# H0: Population1 variance = Population2 variance
# H1: Population1 variance != Population2 variance

# pvalue>0.05: accept null value.
# That means variance are same.
```

# One Sample T Test - Sample and Pop Mean is Given

In [ ]:
```
xbar =
std = np.std(x,ddof=1)
se = stats.sem(mins)
mew =
freedom = len(mins) - 1 # Degrees of Freedom: n-1
alpha = 0.05

# Test Statistic and PValue
teststats, pvalue = st.ttest_1samp(sample, mean) # Sample Data, Pop Mean Par

# Conclusion

if(pvalue>alpha):
    print("Fail to Reject the Ho")
else:
    print("Reject the Ho meaning that the Mean Value is Different from the H
```

## Independent T-Test - Two Samples

In [ ]:
```
# If both samples are independent
teststats, pvalue = stats.ttest_ind(Weight_Male, Weight_Female)

# If both samples are related (pairplot)
teststats, pval = stats.ttest_rel(Marks_before, Marks_after)


```

### If the data is not given:

In [ ]:
```
n1 =  ; xbar1 =  ; sd1 =
n2 =  ; xbar2 =  ; sd2 =
df = n1+n2-2

t = (xbar1 - xbar2)/np.sqrt((sd1**2/n1)+(sd2**2/n2)) ; print(t)

# OR

st.ttest_ind_from_stats(n1,xbar1,sd1,n2,xbar2,sd2)
```

# ANOVA

## One way Anova

In [ ]:
```
# Ho: Tha the Income is Same Across All Gyms
# H1: That the Income is not Same Across All Gyms

stats.f_oneway(Gym_1, Gym_2, Gym_3)

# Since  P Value is > 0.05,  We Fail to Reject the Ho.
```

## Two way Anova

In [ ]:
```
import statsmodels.api as sm
from statsmodels.formula.api import ols

model = ols('M_Income~Gym', data = income).fit() # Ordinary Least Square Met
```

**And**

In [ ]:
```
# First Step: Fit Anova using OLS Method
# Step02: Generate the Anova Table: Sum of Sq, Mean Sq, DF, F Test Statistic

# y~x: y is dependent variable where as ind var is x (3 levels: H, S, Suv)

model = ols('Mean_Pressure~Car_Type', data = pressure).fit() # Fiting the Mo

print(sm.stats.anova_lm(model))
```

**In case of rejecting null hypothesis we recommend to see all the changes happened into the probabilities, we do following.**

In [ ]:
```
from statsmodels.stats.multicomp import pairwise_tukeyhsd
print(pairwise_tukeyhsd(pressure["Mean_Pressure"], pressure["Car_Type"])) #
```

**And**

In [ ]:
```
import statsmodels.api as sm
from statsmodels.formula.api import ols

mobile_model = ols("Qtty~Discount+Loc+Discount:Loc", data = sales).fit()
print(sm.stats.anova_lm(mobile_model))
```

---

# Chi square:

In [ ]:
```
chiteststats = df["Obs-Exp^2"].sum()/Exp_value

crit = stats.chi2.isf(0.01, df) alpha=99, 1-0.99

# If chiteststats > crit, we reject the Ho meaning that the No of Transactio
```

## OR

In [ ]:
```
# One line code:
stats.chisquare(df["Users"], df["Exp_Values"])

# If p_value < alpha, we reject our null hypothesis.
```

## OR

In [ ]:
```
data = pd.crosstab(tips.sex, tips.day)
```

In [ ]:
```
# Contingency Table to generate Statistic
# Test Stat Value, P Value, DF, Exp Value
teststats, pvalue, df, exp_freq = stats.chi2_contingency(data)

print(teststats)
print(pvalue)
print(df) # Degrees of Freedom = (r-1)*(c-1) = (2-1)*(4-1) = 3
print(exp_freq)
```

### If expected value is not given:

In [ ]:
```
st.chisquare(df.Column_name)
```

### OR

In [ ]:
```
df5["Obs"] = np.sum(df5["Obs"])/shape.df5[0]

st.chisquare(df5["Obs"], df5["Exp"]) # df5['Exp'] = Expected column, was cre
```

---

# Machine Learning:

### Simple Linear Regreesion:

In [ ]:
```
st.linregress(df.a, df.b) # a & b are columns of the dataframe
```

In [ ]:
```
from sklearn.linear_model import LinearRegression

LR = LinearRegression()
LR.fit(X, y)

print(f'Coefficients: {lin_reg.coef_}')
print(f'Intercept: {lin_reg.intercept_}')
print(f'R^2 score: {lin_reg.score(X, y)}')
```

## OR

In [ ]:
```
xbar = df.x.mean()
ybar = df.u.mean()

num = np.sum((df.x-xbar)*(df.u-ybar))
deno = np.sum((df.x-xbar)**2)
slope = num/deno

print("Slope: ", slope)

Intercept = ybar - slope * xbar

print("Intercept: ", Intercept)
```

In [ ]:
```
import statsmodels.formula.api as sm

model = sm.ols('Exp~Income', data=food).fit()
model.summary()
```

## OR : A different way of making a model.

In [ ]:
```
import statsmodels.api as sm
from statsmodels.api import OLS
life1 = life[['Hepatitis ','Diphtheria ','Polio','Expected']]
life11 = life1.iloc[:,[0,1,2,3]]
x = life11.drop('Expected', axis=1)
x = sm.add_constant(x)
y = life11.Expected
model = OLS(y,x).fit()
model.summary()
```

**For R2 value:**

In [ ]:
```
model.rsquared
```

**Root mean square & Mean absolute error:**

In [ ]:
```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_scor
RMSE = np.sqrt(mean_squared_error(food.Exp, food.Pred_value))

MAE = mean_absolute_error(food.Exp, food.Pred_value)
```

OR / Complete model.

In [ ]:
```
import statsmodels.api as sm
from statsmodels.api import OLS

x = lungs.drop(['FEV','Smoke','Gender'], axis=1)
x = sm.add_constant(x)
y = lungs.FEV
mod = OLS(y,x).fit()

from sklearn.metrics import mean_absolute_error, mean_squared_error
RMSE = np.sqrt(mean_squared_error(y, mod.predict(x)))
MAE = mean_absolute_error(y, mod.predict(x))

print(RMSE, MAE)
```

OR

In [ ]:
```
# To create constant (intercept into the data)
x = energy2.drop(['Appliances','Press_mm_hg'], axis=1)
y = energy2.Appliances

import statsmodels.api as sm
x = sm.add_constant(x)

from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=0.20, ran

model = sm.OLS(y_train,x_train).fit()
model.summary()
```

## The mean absolute percentage error (MAPE) is commonly used to measure the predictive accuracy of models.

In [ ]:
```
def mape(actual, pred):
    actual, pred = np.array(actual), np.array(pred)
    return np.mean(np.abs((actual - pred) / actual)) * 100
mape(y, mod.predict(x))
```

## One line code for Linear Regression:

In [ ]:
```
def Linear_regression(df,x,y):
    from sklearn.linear_model import LinearRegression
    LR = LinearRegression()
    LR.fit(x,y)
    pred = LR.predict(x)

    from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_
    n = df.shape[0]
    k = df.shape[1]-1
    factor = (n-1)/(n-k-1)

    RMSE= np.sqrt(mean_squared_error(y,pred))
    MAE=mean_absolute_error(y,pred)
    R2=r2_score(y,pred)
    Ajd_R2 = 1 - ((1-R2) * factor)

    df1 = pd.DataFrame({'Data':['RMSE','MAE','R2','Ajd_R2'], 'Values':[RMSE,
    return(df1)

Linear_regression(df=premium,x=premium.drop('Premium',axis=1),y=premium.Prem
```

OR

In [ ]:
```
def Linear_regression(df,x,y):
    from sklearn.linear_model import LinearRegression
    LR = LinearRegression()
    pred = LR.fit(x,y).predict(x)

    from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_
    n = df.shape[0]
    k = df.shape[1]-1
    factor = (n-1)/(n-k-1)

    RMSE= np.sqrt(mean_squared_error(y,pred)); MAE=mean_absolute_error(y,pre
    Ajd_R2 = 1 - ((1-R2) * factor)
    df1 = pd.DataFrame({'Data':['RMSE','MAE','R2','Ajd_R2'], 'Values':[RMSE,
                        index=['RMSE','MAE','R2','Ajd_R2'])

    def analysis():
        global R2
        if R2>=0.5:
            return 'The model is quite good for prediction.'
        else:
            return 'Model is not much appropriate.'

    return(df1[['Values']], analysis())

Linear_regression(df=premium,x=premium.drop('Premium',axis=1),y=premium.Prem
```

## To check the accuracy of the model:

In [ ]:
```
from sklearn.model_selection import train_test_split
xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size = 0.2, rando

lin_reg.score(xtrain, ytrain) , lin_reg.score(xtest, ytest)
```

## Analysis with Random Forest Regression:

In [ ]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_erro
RF = RandomForestRegressor()

pred1 = RF.fit(x,y).predict(x)

print(r2_score(y,pred1), np.sqrt(mean_squared_error(y,pred1)), mean_absolute
```

# BaggingRegressor method of Model building:

In [ ]:
```
from sklearn.ensemble import BaggingRegressor
Bagg = BaggingRegressor()

pred = Bagg.fit(X,y).predict(test_dummies) # test_dummies, is the pd.get_dim

Prediction = solution.copy() # Solution: DataFrame of test_data, for predict

Prediction['Item_Outlet_Sales'] = pred

Prediction.head()
```

## Calculations for Fstat and pvalue

In [ ]:
```
yp = -11.6905 * premium.Mileage + 327.0860 # Coefficient value and Intercept
SSE = np.sum((premium.Premium - yp)**2)
SSR = np.sum((yp-premium.Premium.mean())**2)
n = premium.shape[0]
MSR = SSR/1 ; MSE = SSE / (n-1-1)
Fst = MSR/MSE

1 - st.f.cdf(Fst, 1, n-1-1)
```

OR

In [ ]:
```
tstat = -11.6905/6.006 # Coefficient value and std_error
df = premium.shape[0]-2
pvalue = (1- st.t.cdf(abs(tstat), df))*2
```

## Calculations for Confidence Interval

In [ ]:
```
df = premium.shape[0]-2
alp = 0.025

tcrit = abs(st.t.ppf(alp,df))
b1 = -11.6905 ; se = 6.006

b1 - tcrit*se , b1 + tcrit*se
```

# Assumptions

**Residual plot:**

In [ ]:
```
plt.rcParams['figure.figsize'] = [15,15]

res = model.resid ; fit = model.fittedvalues

sns.residplot(fit, res)
plt.show()
```

**Linearity:**

In [ ]:
```
import statsmodels.api as sm
sm.stats.diagnostic.linear_rainbow(model)
# pvalue > alpha, hence we failed to reject the null hypo. Data is linear.
```

**Probabilty plot:**

In [ ]:
```
plt.rcParams['figure.figsize'] = [15,15]

# Probplot
from scipy.stats import probplot

probplot(model.resid, plot = plt)
plt.show()

# Inferece: Since most of the points are closer to the red line hence, the d
# If the points deviated from the red line, we can conclude that the data is
```

**Normality:**

In [ ]:
```
# Shapiro Wilk Test of Normality: TGT & Residual.

stats.shapiro(model.resid) # Residuals are Normally Distributed.
#stats.shapiro(premium["Premium"]) # Tgt Variable is normally Distributed.
#pvalue>5% then Data is normal.
```

OR

In [ ]:
```
tstat, pvalue = st.jarque_bera(model.resid)
#pvalue>5% then Data is normal.
```

**Multicollinearity:**

In [ ]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor

x = premium.drop('Premium', axis=1)

li = []

for i in range(x.shape[1]):
    li.append(variance_inflation_factor(x.values, i))

pd.DataFrame({'columns':x.columns, 'VIF':li}).sort_values(by='VIF')

# As VIF increases, probability of applying regression gets low.
```

**Autcorrelation:**

In [ ]:
```
from statsmodels.stats.stattools import durbin_watson

durbin_watson(model.resid)

# If errors are autocorrelated: Seems feature is significant but it actually
```

**Thumb Rule - Test Stats is between 1.5 to 2.5 then we consider it to be normal. Values beyond this range is a sign of Autocorrelation**

**Inf: Since it is between 1.5 - 2.5, we fail to reject the Ho.**

**Parameters: DW Test Stat = 2, No Correlation, 0-2: +ve ACorrelation, >2 to 4: -ve ACorrel.**

In [ ]:
```
import statsmodels.tsa.api as smt

acf = smt.graphics.plot_acf(LR.resid, lags=31 , alpha=0.05)
acf.show()
```

**Heteroskedasticity:**

In [ ]:
```
import statsmodels.stats.api as ssa

predictors = premium.drop("Premium", axis =1 )

# residuals, predictors:: endog, exhog
# Generate 04 statistic - Test Stat of BP, PValue(BP), Ftest & Pvalue(Ftest)
teststats, pvalue, ftest, f_pvalue = ssa.het_breuschpagan(model.resid, predi

# pvalue > 0.05, Data is not suffering from heteroskedasticity and there is
```

## Interaction Effect:

In [ ]:
```
premium["Int_Eng_Age"] = premium["Engine"]*premium["Age"]

finalmodel = ols("Premium~Int_Eng_Age+Engine", data = premium).fit()
# OR
finalmodel = ols("Premium~Engine:Age+Engine", data = premium).fit()

finalmodel.summary()
```

# Feature Selection:

In [ ]:
```
np.abs(finalmodel.params[1:]).sort_values().plot(kind = "barh")
# graph will show, which feature is contributing well.
```

**OR**

In [ ]:
```
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor() #Created a Machine

# Input and Output Variable
X = premium.drop(["Mileage","Premium"], axis = 1)
y = premium.Premium

# Fit and Predict

pred = rf.fit(X,y).predict(X)

# To look the data:
pd.DataFrame({"Features":rf.feature_importances_},
             index = X.columns)

# IN RF, We already have a parameter called feature importance
# Plot will show a good idea of best features.
pd.DataFrame({"Features":rf.feature_importances_},
             index = X.columns).sort_values(by = "Features").plot(kind = "ba
```

In [ ]:
```
# Calc the R2 score and RMSE
from sklearn.metrics import r2_score, mean_squared_error

#print("LR R2Score: ", r2_score(y, finalmodel.predict(premium.drop("Mileage"
print("RF R2Score: ", r2_score(y, pred))

#print("LR rmse: ", np.sqrt(mean_squared_error(y, finalmodel.predict(X))))
print("RF rmse: ", np.sqrt(mean_squared_error(y, pred)))
```

## Recursive Feature Elimination:

In [ ]:
```
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor()

from sklearn.feature_selection import RFE
rfe = RFE(estimator=RandomForestRegressor(),
          n_features_to_select=2) # estimator-Model, n_features_to_select

newinput = rfe.fit_transform(X,y) # return the dataset with the correct set

# Finding the Column Names
colnos = rfe.get_support(indices = True) # returns the indices or the column

X.iloc[:, colnos].columns # this is the correct approach
```

## OR

In [ ]:
```
from sklearn.feature_selection import RFE
from sklearn.linear_model import LinearRegression
lr = LinearRegression()

pred_lr = lr.fit(X,y).predict(X)

rfe = RFE(estimator = LinearRegression(),
          n_features_to_select=2)

newinput = rfe.fit(X,y)
colnos = rfe.get_support(indices = True)
print(X.iloc[:, colnos].columns)
```

## OR

## Sequencial Feature Selection:

In [ ]:
```
#pip install mlxtend - Sequential Feature Selector
from mlxtend.feature_selection import SequentialFeatureSelector as sfs

from sklearn.linear_model import LinearRegression
lr = LinearRegression()

fs = sfs(estimator=LinearRegression(), k_features="best", forward = True,
         verbose = 2, scoring = "r2", cv = 14)

# fit the model and generate the feature names
X = premium.drop(["Mileage","Premium"], axis = 1)
y = premium.Premium

sfsmodel = fs.fit(X, y)

# Generate the feature names
print("Features: ", sfsmodel.k_feature_names_)
print("R2 Score: ", sfsmodel.k_score_)
```

# To get more precide model we do boosting:

In [ ]:
```
from sklearn.ensemble import GradientBoostingRegressor
gbm = GradientBoostingRegressor()

from mlxtend.feature_selection import SequentialFeatureSelector as sfs

from sklearn.linear_model import LinearRegression
lr = LinearRegression()

fs = sfs(estimator=gbm, k_features="best", forward = True,
         verbose = 2, scoring = "r2", cv = 14)

# fit the model and generate the feature names
X = premium.drop(["Mileage","Premium"], axis = 1)
y = premium.Premium

sfsmodel = fs.fit(X, y)

# Generate the feature names
print("Features: ", sfsmodel.k_feature_names_)
print("R2 Score: ", sfsmodel.k_score_)
```

## Feature selector Estimator

In [ ]:
```
SFS = sfs(estimator = ls, verbose=0, k_features = "best",
    cv = 5, scoring = "r2")

sfsmodel = SFS.fit(x_train,y_train)
print("Features: ", sfsmodel.k_feature_names_)
print("R Squared: ", sfsmodel.k_score_)
```

## Cross Validation:

### K Fold:

In [ ]:
```
from sklearn.model_selection import KFold
n=10
KF = KFold(n_splits=n, shuffle=True)

x = energy2.drop(["Appliances", "lights"], axis = 1)
y = energy2.Appliances

for i in range(n):
    result = next(KF.split(x))
    x_train = x.iloc[result[0]]
    x_test = x.iloc[result[1]]
    y_train = y.iloc[result[0]]
    y_test = y.iloc[result[1]]

x_train.shape, x_test.shape, y_train.shape, y_test.shape
```

### One line code for cross validation score:

In [ ]:
```
from sklearn.model_selection import cross_val_score

from sklearn.model_selection import KFold
n =5

# shuffle the data after every draw
# n splits - how many splits

#kf_ = KFold(n_splits=n, shuffle = True)
cross_val_score(estimator=lrfit, X = x_train, y = y_train, scoring = "r2", c
```

### OR

In [ ]:
```
from sklearn.model_selection import cross_val_score

from sklearn.model_selection import KFold
n =5

x = df.drop('quality', axis=1)
y = df.quality

lrfit = LinearRegression()

cross_val_score(estimator=lrfit, X=x, y= y, scoring = "r2", cv = 2).mean()
```

## To extract numerical columns only

In [ ]:
```
df.select_dtypes(np.numeric)
```

### Overfitting and Underfitting

In [ ]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

x = fish.drop(['Weight','Species'], axis = 1)
y = fish['Weight']
xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size = 0.2, rando

from sklearn.linear_model import LinearRegression
lrmodel = LinearRegression()

pred_train = lrmodel.fit(xtrain, ytrain).predict(xtrain)
pred_test = lrmodel.fit(xtrain, ytrain).predict(xtest)

print("RMSE for Train: ",np.sqrt(mean_squared_error(ytrain, pred_train)))
print("RMSE for Test: ",np.sqrt(mean_squared_error(ytest, pred_test)))
```

## Elastic regression:

In [ ]:
```
from numpy import mean
from numpy import std
from numpy import absolute
from pandas import read_csv
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedKFold
from sklearn.linear_model import ElasticNet

model = ElasticNet(alpha=1.0, l1_ratio=0.001)
# define model evaluation method
cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)
# evaluate model
scores = cross_val_score(model, x, y, scoring='neg_mean_absolute_error', cv=
# force scores to be positive
scores = absolute(scores)
print('Mean MAE: %.3f (%.3f)' % (mean(scores), std(scores)))
```

## GridSearch

In [ ]:
```
from sklearn.model_selection import GridSearchCV
alphas = [{'alpha':[0.1, 0.01, 0.02, 0.0005, 0.09, 0.001, 0.5, 0.99, 1, 2, 5
grid =GridSearchCV(estimator = Ridge(normalize=True), param_grid= alphas, cv

grid.fit(xtrain,ytrain)

grid.best_params_
```

OR

In [ ]:
```
params = {'alpha' : [0.1,0.5,1,1.5,2,2.5,3,3.5,4] ,'l1_ratio' : [0.5,0.55,0.

from sklearn.model_selection import GridSearchCV

grid = GridSearchCV(ene, param_grid=params,cv=10)
grid.fit(xtrain,ytrain)

grid.best_params_
```

## Lasso & Regression

In [ ]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LassoCV, RidgeCV, ElasticNet
```

## BaggingRegressor

In [ ]:
```
from sklearn.ensemble import BaggingRegressor
Bagg = BaggingRegressor()

pred = Bagg.fit(X,y).predict(test_dummies)

Prediction = solution.copy()

Prediction['Item_Outlet_Sales'] = pred

Prediction.head()
```

In [ ]:
```
Prediction.to_csv('Predictions.csv', index=False)
```

Awesome...

# END