

FUNCTIONS AND WAYS OF HANDLING DATA

Importing and Exporting data:

Importing and exporting data in python is done by using pandas library

Import pandas as pd

```
df = pd.read_csv("path of file")
```

```
pd.to_csv("path of new file and location")
```

Getting insights into data

To get insights into data we can use

```
df.head() //to see first 10 rows
```

```
df.tail() //to see last 10 rows
```

```
df.describe() //to get insight into data as average and other mathematical factors.
```

```
df.describe(include="all") //for all data
```

```
df.info() //can also be used to get info about top 30 and bottom 30 rows.
```

```
Df.shape //will tell you about the shape of the data.
```

Connecting to the database

```
from dbmodule import connect

#Create connection object
connection = connect('databasename','username','pswd')

#Create a cursor object
cursor = connection.cursor()

#Run queries
cursor.execute('select * from mytable')
results = cursor.fetchall()

#Free resources
Cursor.close()
connection.close()
```

Cleaning Data, Removing and replacing missing Values

To remove NA values we use

```
df.dropna(subset = “[column_name]”, axis = 0, inplace = True)
```

Axis 0 — x axis – means dropping rows

Axis 1 — y axis – means dropping columns

Inplace = true —means keep the other data intact and just drop the column or row

For replacing values

```
df.replace(“old val”, “new val”) // new val can be anything (mean,median,mode)
```

Simple calculations can be easily done on data frames

Df[“column name”] = df[“column name”]*255

Also we can change column name using

df.rename(columns = {"original": "new"}, inplace= true)

To check datatype of column

df.dtypes()

To change datatype of column we use

df.astype(“data type”)

Numpy linspace

Numpy linspace gives you equally spaced integers in a specified range

`np.linspace(start, end, “HOW MANY NUMBERS YOU WANT”)`

```
bins = np.linspace(min(df[“price”]), max(df[“price”]), 4)
group_names = [“Low”, “Medium”, “High”]
df[“price-binned”] = pd.cut(df[“price”], bins, labels=group_names, include_lowest=True )
```

`pd.cut()` is used to divide the data as per the bins and `labels` is used to label them after dividing.

Turning categorical variables into numerical variables



fuel	gas	diesel
gas	1	0
diesel	0	1
gas	1	0
gas	1	0

```
pd.get_dummies(df['fuel'])
```

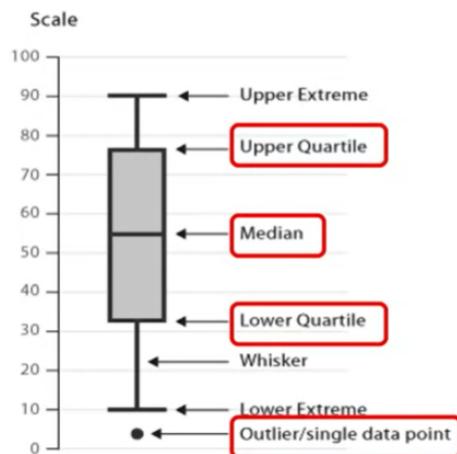
Pd.get_dummies will create dummy variables in the dataset with values as 0 and 1 as false and true respectively.

VALUE COUNTS is used to count the frequency

```
drive_wheels_counts=df["drive-wheels"].value_counts().to_frame()  
drive_wheels_counts.rename(columns={'drive-wheels':'value_counts'}, inplace=True)  
drive_wheels_counts
```

BOX PLOTS --- DATA VISUALIZATION

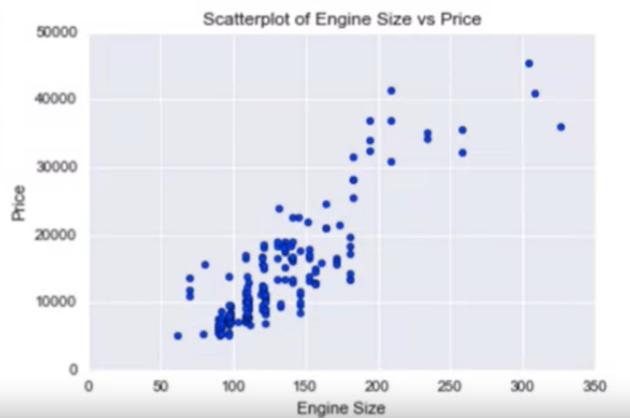
Descriptive Statistics - Box Plots



```
sns.boxplot(x = 'waterfront', y = 'price', data = df)
```

SCATTER PLOTS --- DATA VISUALIZATION

```
y=df[ "price" ]  
x=df[ "engine-size" ]  
plt.scatter(x,y)  
  
plt.title("Scatterplot of Engine Size vs Price")  
plt.xlabel("Engine Size")  
plt.ylabel("Price")
```



GROUP BY

Group by is used to group data into categorical variables.

Groupby()- Example

```
df_test = df[['drive-wheels', 'body-style', 'price']]  
df_grp = df_test.groupby(['drive-wheels', 'body-style'], as_index=False).mean()  
df_grp
```



	drive-wheels	body-style	price
0	4wd	convertible	20239.229524
1	4wd	sedan	12647.333333
2	4wd	wagon	9095.750000
3	fwd	convertible	11595.000000
4	fwd	hardtop	8249.000000
5	fwd	hatchback	8396.387755
6	fwd	sedan	9811.800000
7	fwd	wagon	9997.333333
8	rwd	convertible	23949.600000
9	rwd	hardtop	24202.714286
10	rwd	hatchback	14337.777778
11	rwd	sedan	21711.833333

PIVOT TO BEAUTIFY GROUP BY

Pandas method - Pivot()

- One variable displayed along the columns and the other variable displayed along the rows.

```
df_pivot = df_grp.pivot(index= 'drive-wheels', columns='body-style')
```

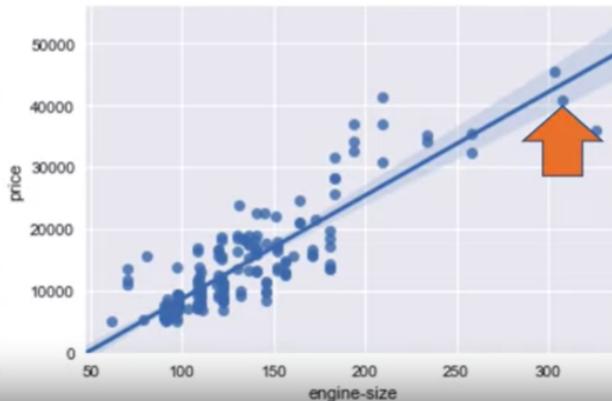
	price					
body-style	convertible	hardtop	hatchback	sedan	wagon	
drive-wheels						
4wd	20239.229524	20239.229524	7603.000000	12647.333333	9095.750000	
fwd	11595.000000	8249.000000	8396.387755	9811.800000	9997.333333	
rwd	23949.600000	24202.714286	14337.777778	21711.833333	16994.222222	

SEABORN IS MORE STATISTICAL THAN MATPLOTLIB

Correlation - Positive Linear Relationship

- Correlation between two features (engine-size and price).

```
sns.regplot(x="engine-size", y= "price", data=df)  
plt.ylim(0, )
```



Sns.regplot() is used in the same manner as scatter plot and gives more insight into data by drawing the slope lines.

PEARSON CORRELATION COEFFICIENT

df.corr()><https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.corr.html>

Pearson Correlation

- Measure the strength of the correlation between two features.
 - Correlation coefficient
 - P-value
- Correlation coefficient
 - Close to +1: Large Positive relationship
 - Close to -1: Large Negative relationship
 - Close to 0: No relationship
- P-value
 - P-value < 0.001 **Strong** certainty in the result
 - P-value < 0.05 **Moderate** certainty in the result
 - P-value < 0.1 **Weak** certainty in the result
 - P-value > 0.1 **No** certainty in the result

```
pearson_coef, p_value = stats.pearsonr(df['horsepower'], df[ 'price'])
```

- Pearson correlation: 0.81
- P-value : 9.35 e-48

MODEL DEVELOPMENT

Linear Regression Model

```
from sklearn.linear_model import LinearRegression  
  
lm=LinearRegression()  
  
X = df[['highway-mpg']]  
  
Y = df['price']  
  
lm.fit(X, Y)  
  
Yhat=lm.predict(X)
```

DISTRIBUTION PLOT

Distribution Plots

```
import seaborn as sns  
  
↓  
  
ax1 = sns.distplot(df['price'], hist=False, color="r", label="Actual Value")  
  
sns.distplot(Yhat, hist=False, color="b", label="Fitted Values" , ax=ax1)
```

POLYNOMIAL DISTRIBUTION

See sometimes linear regression isn't able to fit all the data points correctly here we require polynomials modes.

Polynomial Regression

1. Calculate Polynomial of 3rd order

```
f=np.polyfit(x,y,3)  
p=np.poly1d(f)
```

2. We can print out the model

```
print (p)
```

$$-1.557(x_1)^3 + 204.8(x_1)^2 + 8965 x_1 + 1.37 \times 10^5$$

As the linear regression is not very suitable with the polynomials we use sklearn preprocessing library.

- The "preprocessing" library in scikit-learn,

```
from sklearn.preprocessing import PolynomialFeatures  
pr=PolynomialFeatures(degree=2, include_bias=False)  
x_polly=pr.fit_transform(x[['horsepower', 'curb-weight']])
```

We can normalize each feature using

- For example we can Normalize the each feature simultaneously

```
from sklearn.preprocessing import StandardScaler  
SCALE=StandardScaler()  
SCALE.fit(x_data[['horsepower', 'highway-mpg']])  
x_scale=SCALE.transform(x_data[['horsepower', 'highway-mpg']])
```

PIPELINE

Pipelines are very important in machine learning as they reduce your task of applying separate models one by one.

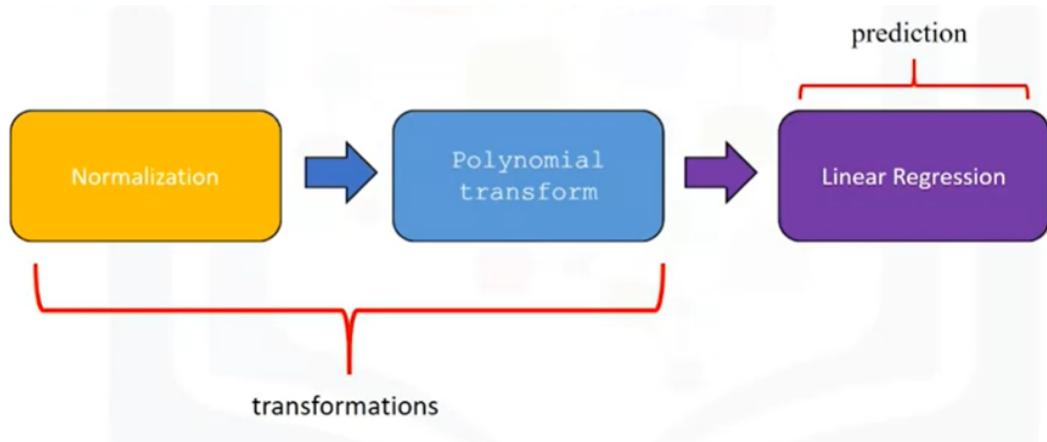
Instead you can create a single pipeline that goes through each model and produces a result.

HOW?

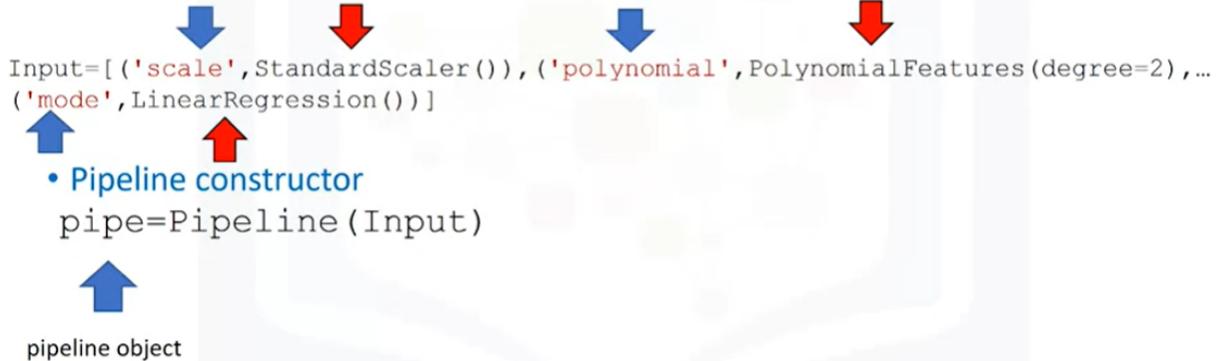
First we need to import the libraries required

```
from sklearn.preprocessing import PolynomialFeatures  
from sklearn.linear_model import LinearRegression  
from sklearn.preprocessing import StandardScaler  
from sklearn.pipeline import Pipeline
```

Here the three processes are combined



Creating a pipeline object



Now after creating a pipeline object we can train and predict using this model

```
Pipe.fit(df[['horsepower', 'curb-weight', 'engine-size', 'highway-mpg']],y)  
yhat=Pipe.predict(X[['horsepower', 'curb-weight', 'engine-size', 'highway-mpg']])
```

Mean Squared Approach and R squared approach?

Train Test Split

Function train_test_split()

- Split data into random train and test subsets

```
from sklearn.model_selection import train_test_split
```

```
x_train, x_test, y_train, y_test = train_test_split(x_data, y_data, test_size=0.3, random_state=0)
```

- **x_data**: features or independent variables
- **y_data**: dataset target: df['price']
- **x_train, y_train**: parts of available data as training set
- **x_test, y_test**: parts of available data as testing set
- **test_size**: percentage of the data for testing (here 30%)
- **random_state**: number generator used for random sampling

CROSS VALIDATION

Cross-validation is a technique for evaluating ML models by training several ML models on subsets of the available input data and evaluating them on the complementary subset of the data. Use cross-validation to detect overfitting, ie, failing to generalize a pattern.

```
from sklearn.model_selection import cross_val_score  
scores = cross_val_score(lr, x_data, y_data, cv=10)
```

cv is used to divide the dataset into parts here cv=10 means dividing the dataset into 10 parts.

Lr – linear regression model.

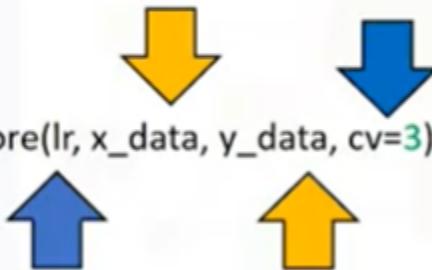
X_data is the data/independent variables.

Y_data is the dependent variable or the target variable.

```
from sklearn.model_selection import cross_val_score
```

```
scores= cross_val_score(lr, x_data, y_data, cv=3)
```

```
np.mean(scores)
```



RIDGE REGRESSION

Ridge Regression

```
from sklearn.linear_model import Ridge
```

```
RidgeModel=Ridge(alpha=0.1)
```

```
RidgeModel.fit(X,y)
```

```
Yhat=RidgeModel.predict(X)
```

GRID SEARCH

```
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV

parameters1= [{"alpha": [0.001,0.1,1, 10, 100, 1000,10000,100000,1000000]}]

RR=Ridge()

Grid1 = GridSearchCV(RR, parameters1,cv=4)

Grid1.fit(x_data[['horsepower', 'curb-weight', 'engine-size', 'highway-mpg']],y_data)

Grid1.best_estimator_

scores = Grid1.cv_results_
scores['mean_test_score']
```