Python Cheat Sheet for Data Analysis

Data Loading

Read CSV dataset

```
# load without header
df = pd.read_csv(<CSV path>, header = None)
# load using first row as header
df = pd.read_csv(<CSV path>, header = 0)
```

Print first few entries

```
#n=number of entries; default 5
df.head(n)
```

Print last few entries

```
#n=number of entries; default 5
df.tail(n)
```

Assign header names

```
df.columns = headers
```

Replace "?" with NaN

```
df = df.replace("?", np.nan)
```

Retrieve data types

```
df.dtypes
```

Retrieve statistical description

```
# default use
df.describe()
# include all attributes
df.describe(include="all")
```

Retrieve data set summary

```
df.info()
```

Save data frame to csv

```
df.to_csv(<output CSV path>)
```

Data Wrangling

Replace missing data with frequency

```
MostFrequentEntry =
df['attribute_name'].value_counts().idxmax()

df['attribute_name'].replace(np.nan,MostFrequentEntry
, inplace=True)
```

Replace missing data with mean

```
AverageValue=
df['attribute'].astype(<data_type>).mean(axis=0)

df['attribute'].replace(np.nan, AverageValue,
inplace=True)
```

Fix the data types

```
df[['attribute1', 'attribute2', ...]] =
df[['attribute1', 'attribute2',
...]].astype('data_type')
#data_type can be int, float, char, etc.
```

Data normalization

```
df['attribute_name'] =
df['attribute_name']/df['attribute_name'].max()
```

Binning

```
bins = np.linspace(min(df['attribute_name']),
max(df['attribute_name'],n)
# n is the number of bins needed

GroupNames = ['Group1','Group2','Group3',...]

df['binned_attribute_name'] =
pd.cut(df['attribute_name'], bins, labels=GroupNames,
include_lowest=True)
```

Change column name

```
df.rename(columns={'old_name':'new_name'},
inplace=True)
```

Indicator variables

```
dummy_variable = pd.get_dummies(df['attribute_name'])

df = pd.concat([df, dummy_variable],axis = 1)
```

Exploratory Data Analysis

Complete data frame correlation

```
df.corr()
```

Specific attribute correlation

```
df[['attribute1','attribute2',...]].corr()
```

Scatter plot

```
from matlplotlib import pyplot as plt
plt.scatter(df[['attribute_1']], df[['attribute_2']])
```

Regression plot

```
import seaborn as sns
sns.regplot(x='attribute_1',y='attribute_2', data=df)
```

Box plot

```
import seaborn as sns
sns.boxplot(x='attribute_1',y='attribute_2', data=df)
```

Grouping by attributes

```
df_group = df[['attribute_1','attribute_2',...]]
```

GroupBy statements

```
# Group by a single attribute
df_group = df_group.groupby(['attribute_1'],
as_index=False).mean()

# Group by multiple attributes
df_group = df_group.groupby(['attribute_1',
'attribute_2'],as_index=False).mean()
```

Pivot tables

```
grouped_pivot =
df_group.pivot(index='attribute_1',columns='attribute
_2')
```

Pseudocolor plot

```
from matlplotlib import pyplot as plt
plt.pcolor(grouped_pivot, cmap='RdBu')
```

Pearson Coefficient and p-value

```
from scipy import stats
pearson_coef,p_value=stats.pearsonr(df['attribute_1']
, df['attribute_2'])
```

Skills Network

Python Cheat Sheet for Data Analysis

Model Development

Linear regression

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

Train linear regression model

```
X = df[['attribute_1', 'attribute_2', ...]]
Y = df['target_attribute']
lr.fit(X,Y)
```

Generate output predictions

```
Y_hat = lr.predict(X)
```

Identify the coefficient and intercept

```
coeff = lr.coef_
intercept = lr.intercept_
```

Residual plot

```
import seaborn as sns
sns.residplot(x=df[['attribute_1']],
y=df[['attribute_2']])
```

Distribution plot

```
import seaborn as sns
sns.distplot(df['attribute_name'], hist=False)
# can include other parameters like color, label,
etc.
```

Polynomial regression

```
f = np.polyfit(x, y, n)
#creates the polynomial features of order n

p = np.poly1d(f)
#p becomes the polynomial model used to generate the
predicted output

Y_hat = p(x)
# Y_hat is the predicted output
```

Multi-variate polynomial regression

```
from sklearn.preprocessing import PolynomialFeatures

Z = df[['attribute_1','attribute_2',...]]
pr=PolynomialFeatures(degree=n)
Z_pr=pr.fit_transform(Z)
```

Pipeline

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
Input=[('scale',StandardScaler()), ('polynomial',
PolynomialFeatures(include_bias=False)),
('model',LinearRegression())]

pipe=Pipeline(Input)

Z = Z.astype(float)
pipe.fit(Z,y)
ypipe=pipe.predict(Z)
```

R^2 value

```
# For linear regression model
X = df[['attribute_1', 'attribute_2', ...]]
Y = df['target_attribute']

lr.fit(X,Y)
R2_score = lr.score(X,Y)

# For polynomial regression model
from sklearn.metrics import r2_score

f = np.polyfit(x, y, n)
p = np.poly1d(f)
R2_score = r2_score(y, p(x))
```

MSE value

```
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(Y, Yhat)
```

Model Evaluation and Refinement

Split data for training and testing

```
from sklearn.model_selection import train_test_split

y_data = df['target_attribute']
x_data=df.drop('target_attribute',axis=1)

x_train, x_test, y_train, y_test =
train_test_split(x_data, y_data, test_size=0.10,
random_state=1)
```

Cross-validation score

```
from sklearn.model_selection import cross_val_score

from sklearn.linear_model import LinearRegression

lre=LinearRegression()

Rcross =
cross_val_score(lre,x_data[['attribute_1']],y_data,cv
=n)
# n indicates number of times, or folds, for which
the cross validation is to be done

Mean = Rcross.mean()
Std_dev = Rcross.std()
```

Cross-validation prediction

```
from sklearn.model_selection import cross_val_score

from sklearn.linear_model import LinearRegression

lre=LinearRegression()

yhat = cross_val_predict(lre,x_data[['attribute_1']],
y_data,cv=4)
```

Ridge regression and prediction

```
from sklearn.linear_model import Ridge
pr=PolynomialFeatures(degree=2)

x_train_pr=pr.fit_transform(x_train[['attribute_1',
'attribute_2', ...]])

x_test_pr=pr.fit_transform(x_test[['attribute_1',
'attribute_2', ...]])

RidgeModel=Ridge(alpha=1)
RidgeModel.fit(x_train_pr, y_train)
yhat = RigeModel.predict(x_test_pr)
```

Grid search

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

parameters= [{'alpha': [0.001,0.1,1, 10, 100, 1000,
10000, ...]}]

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

Grid1.fit(x_data[['attribute_1', 'attribute_2',
...]], y_data)

BestRR=Grid1.best_estimator_

BestRR.score(x_test[['attribute_1', 'attribute_2',
...]], y_te
```