PRIME INTUIT
Finishing School

Pandas – Data Frame Objects

DataFrame is a 2-dimensional labeled data structure with columns of potentially different types. You can think of it like a spreadsheet or SQL table, or a dict of Series objects.

PRIME INTUIT
Finishing School

Pandas – Data Frame Objects

DataFrame is a 2-dimensional labeled data structure with columns of potentially different types. You can think of it like a spreadsheet or SQL table, or a dict of Series objects.

PRIME INTUIT
Finishing School

Pandas – Data Frame Objects

```
Import Numpy as np
Import Pandas as pd

#Creating Dataframe Objects

Arr = np.random.randint(0, 10, (5, 3))
Arr

Df = pd.DataFrame(arr)
Df
Df.values
Df.index
Df.columns
Df.values[0]
```

PRIME INTUIT
Finishing School

Pandas – Data Frame Objects

Df.index =[‘R1’, ‘R2’, ‘R3’, ‘R4’, ‘R5’]
Df.columns = [c1, c2, c3]
DF

Implicitly obtain the appropriate locks for your application at the point at which they are needed. An operation that reads an object will obtain a read lock; an operation that modifies an object will obtain a write lock.

Explicit Lock: - Lock is explicitly requested for a record or table.
Implicit Lock: - Lock is implied but is not acquired

Df.loc[R3, C2]
Df.iloc[2, 1]
Df.iloc[2:4, 1:3]

PRIME INTUIT
Finishing School

Pandas – Data Frame Objects

Df.loc =['R3': 'R5', 'c2' : 'c3']

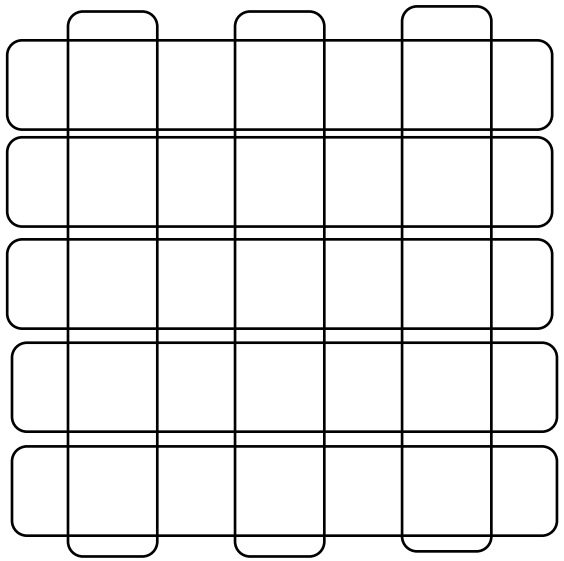
Df.iloc[0]

Type(df.iloc[0]) # you can think of df as a collection of series in both x and y axis

Df.iloc[:, 0]

Df.shape

Df.T # Transpose



PRIME INTUIT
Finishing School

Pandas – Data Frame Objects

```
#  Write a function to create data frame with n rows and n columns

Def create_df(nrows, ncols, maxrand = 10):
        arr = np.random.randint(0, maxrand, (nrows, ncols))
        df = pd.DataFrame(arr)
        df.index = [‘R’ + str(x) for x in np.arange(1, nrows + 1)]
        df.coloumns = [‘C’ + str(x) for x in np.arange(1, ncols + 1)]
        return df

Create_df(5, 3)

Create_df(2, 5)
```

```
[27] def create_df(nRows, nCols, maxRand=10):
        arr = np.random.randint(0, maxRand, (nRows, nCols))
        df = pd.DataFrame(arr)
        df.index = ['R' + str(x) for x in np.arange(1, nRows+1)]
        df.columns = ['C' + str(x) for x in np.arange(1, nCols+1)]
        return df
```

PRIME INTUIT
Finishing School

Pandas – Data Frame Objects

Creating a Data Frame using multiple Series

```
mass1 = pd.Series((0.33, 4.07, 5.97, 0.642, 1090, 568, 86.0, 102, 0.0146, 0.000292), index = ['mercury', 'v
enus', 'earth', 'mars', 'jupiter', 'saturn', 'uranus', 'neptune', 'pluto', 'eris'])
#print(mass1)
dia1 = pd.Series((4079, 12104, 12756, 3475, 6792, 142904, 120536, 51110, 49528, 2370, 2326), index = ['
mercury', 'venus', 'earth', 'moon', 'mars', 'jupiter', 'saturn', 'uranus', 'neptune', 'pluto', 'eris'])
#print(dia1)
```

Df = pd.DataFrame({‘mass’ : mass1, ‘diameter’, dia1})
Df

df[‘mass’]
Df[‘diameter’]

Df[‘earth’]

PRIME INTUIT
Finishing School

Pandas – Data Frame Objects

```
#  Creating a Data Frame using multiple Series

Df['mass']['earth']

Df.mass.earth

#  Creating a new coloumn

Df['pop'] = 0

Df

Df['pop']['earth'] =8000000000

df
```

PRIME INTUIT
Finishing School

Pandas – Data Frame Objects

```
#  Creating a Data Frame using multiple Series

Df['mass'] == df.mass

df['mass'] is df.mass

Df[pop] is df.pop

Df.pop                    # pop function pops an item from the data frame

Df.loc ['earth']

Df.loc[:, mass]
```

PRIME INTUIT
Finishing School

Pandas – Data Frame Objects

```
#  Creating a new row
Df.loc['mean'] = 0

Df

Df.drop('mean')
Df.drop('pop', axis =1)
df
Df.drop('mean', inplace = True)
Df.drop('pop', axis =1, inplace = True)
df

Np.mean(df['mass'])
Df.loc['mean'] = [np.mean(df['mass]), np.mean(df['diameter])]
df
```

Pandas – Data Frame Objects

```
# Creating a function to create a new row

Def create_mean_row(df):
        df.loc['col_mean'] = [np.mean(df[col1]) for col in df.columns]
        return df

Create_mean_row(df)

Def create_mean_row(df):
        df.loc['col_mean1'] = df.mean()
        return df

Create_mean_row(df)

Df = create_df(5, 3)
Df
Df.mean()
```

PRIME INTUIT
Finishing School

Pandas – Data Frame Objects

```
#  Creating a function to create a new row
# what if I want to compute mean for the row:

Df.mean(axis =1)

Df['row_mean'] = df.mean(axis =1)
Df

Df.loc['col_mean'] = df.mean()
Df

Df.median()
Df.min()
Df.max()
Df.quantile(0.25)
Df.drop('row_mean', axis =1)
```

PRIME INTUIT
Finishing School

Pandas – Data Frame Objects

```
# Other ways of computing the details

Df.describe()
mass1 = pd.Series((0.33, 4.07, 5.97, 0.642, 1090, 568, 86.0, 102, 0.0146, 0.0002
92), index = ['mercury', 'venus', 'earth', 'mars', 'jupiter', 'saturn', 'uranus', 'nept
une', 'pluto', 'eris'])
#print(mass1)
dia1 = pd.Series((4079, 12104, 12756, 3475, 6792, 142904, 120536, 51110, 495
28, 2370, 2326), index = ['mercury', 'venus', 'earth', 'moon', 'mars', 'jupiter', 'sat
urn', 'uranus', 'neptune', 'pluto', 'eris'])
#print(dia1)


Planets = pd.DataFrame({‘mass’ : mass1, ‘diameter’, dia1})
Planets.describe()
```

PRIME INTUIT
Finishing School

Pandas – Data Frame Objects

Seaborn – Planets data set

Import seaborn as sns

Df = sns.load_dataset('planets')

df.info()

Df.head()

Df.tail()

Df.describe()

PRIME INTUIT
Finishing School

Pandas – Data Frame Objects

```
# Task 1: Go through each row of the df and delete (drop) if any of the cols is null

# Regular way

For r in df.index:
        for c in df.columns:
                if pd.isnull(df.loc[r, c]):
                df.drop(r, inplace = True)
                break


Df.describe()

For I, r in df.iterrows():
        print(i)
        print(r)
        break
```

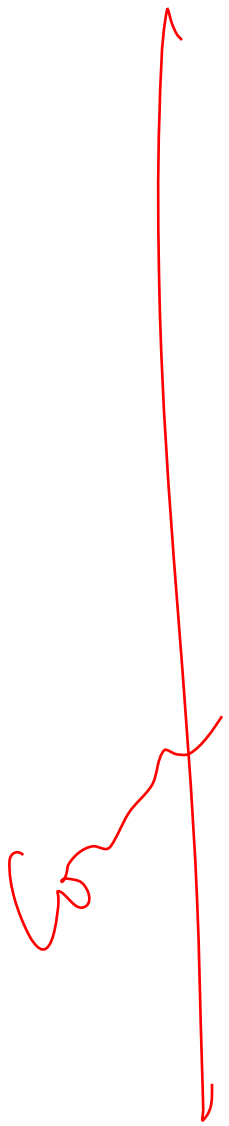
PRIME INTUIT
Finishing School

Pandas – Data Frame Objects

```
# Task 1: Go through each row of the df and delete (drop) if any of the cols is null

For I, r in df.iterrows():
        print(pd.isnull(r))
        break

For I, r in df.iterrows():
        print(pd.isnull(r).any()):
        break

For I, r in df. Iterrows():
        if pd.isnull(r).any():
        df.drop(I, inplace = True)
```

PRIME INTUIT
Finishing School

Pandas – Data Frame Objects



```
Df.describe()

# most efficient way

Df.dropna(inplace = true)
Df.describe()

# Task 2: filter rows for planets found in 2010s and methods is ‘Radial Velocity’ &
‘Transit’ & distance is > 75%
```

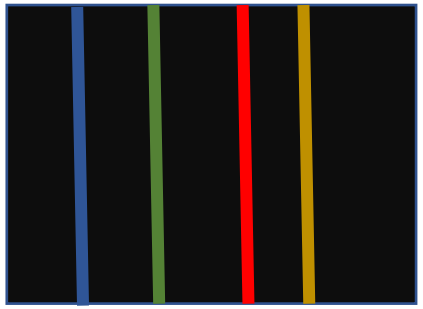
PRIME INTUIT
Finishing School

Pandas – Data Frame Objects

```
# Method: 1

Df_1 = df.copy()
Per_75 = df.distance.quantile(0.75)
For I, r in df_1.iterrows():
        if r['year'] < 2010:
                df_1.drop(I, inplace = True)
                continue
        if r['method'] != 'Radial Velocity' and r['method'] != 'Transit':
                df_1.drop(I, inplace = True)
                continue
        if r['distance'] < per_75:
                df_1.drop(I, inplace = True)
                continue
Df_1.describe()
df_1.head()
Per_75
Df_1.tail()
```

PRIME INTUIT
Finishing School

Pandas – Data Frame Objects



```
# Method: 2 using indexing with conditions

Df_2 = df.copy()
Df_3 = df_2[
            (df_2['year'] > = 2010) &
            ((df_2['method'] == 'Radial Velocity') | (df_2['method'] == 'Transit')) &
            (df_2['distance'] > per_75)
]

Df.describe()
```

Task 3: Modify the method column to have only the abbreviation of each method

PRIME INTUIT
Finishing School

Pandas – Data Frame Objects

```
# Method: 1

Df_2.method.unique()

S = 'Radial Velocity'
S.split(' ')
[x[0] for x in S.split(' ')]
' '.join([x[0] for x in S.split(' ')])

Short_Names = {}
For s in df.method.unique():
        short_name[s] = ' '.join([x[0] for x in S.split(' ')])

Print(Short_ Name)
```

PRIME INTUIT
Finishing School

Pandas – Data Frame Objects

```
# Method: 1 Continued……..

For I, r in df.iterrows():
        df.loc[I, ‘short_method’] = Short_Names.get(r[‘method’], r[‘method’])
        # new coloumn          Get names form short_names dict by using get function

Df.head()

Df.tail()
```

PRIME INTUIT
Finishing School

Pandas – Data Frame Objects

```
# Method: 2

Df5 = sns.load_dataset('planets')
Def Shorten_method(s):
        Short_Names.get(s,s)

df5['Short_Method'] = df['method'].apply(Shorten_method)

#Apply Shorten_method to each coloumn in df5/method column take the result and put it
into Short_Method column.
```

```
# Task 4: count the number of planets discovered for each method type
```

PRIME INTUIT
Finishing School

Pandas – Data Frame Objects

 # Task 4: count the number of planets discovered for each method type

 # Step 1. Split the data frame into smaller chunks (in this case they should have the same method name)

 # Step 2. Apply some function in each smaller chuck (count function)

 # Step 3. Aggregate the results from each chunk together.

PRIME INTUIT
Finishing School

Pandas – Data Frame Objects

```
# Method 1


D = {}      # for printing
For m in df5.method.unique():
            df5[df5.method == m]                    # aggregation
            df5[df5.method == m].count()            # count
            print(m)
            print(df5[df5.method == m].count())          # print all the coloums

             print(df5[df5.method == m]['method'].count())       # aggregation
            d[m] = df[df.method == m]['method'].count()
print(d)
```

PRIME INTUIT
Finishing School

Pandas – Data Frame Objects

```
# Method 2


~~Df5.groupby('method').count()~~
# grouping based on column 'method'

Df5.groupby('method')['method'].count()
# grouping based on column 'method' and count on column 'method'
Print(d)

Df5.groupby('method') ['distance'].mean()
```

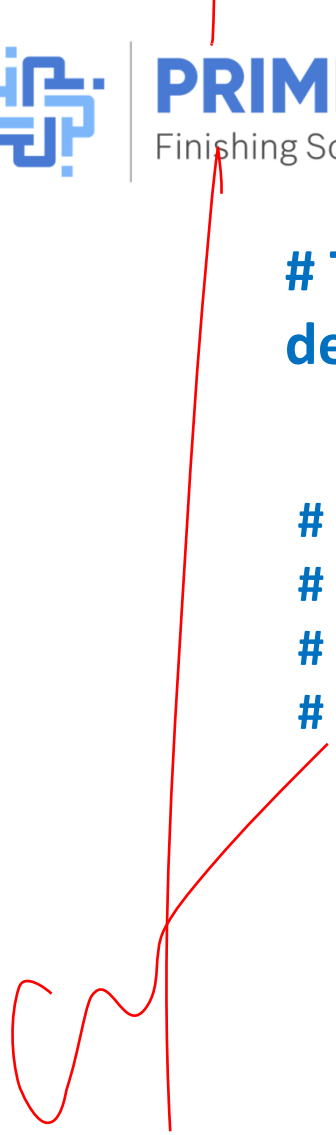
```
# Task 5: Find out what fraction of planets have been found in the last decase(2010s) across each method type
```

PRIME INTUIT
Finishing School

Pandas – Data Frame Objects

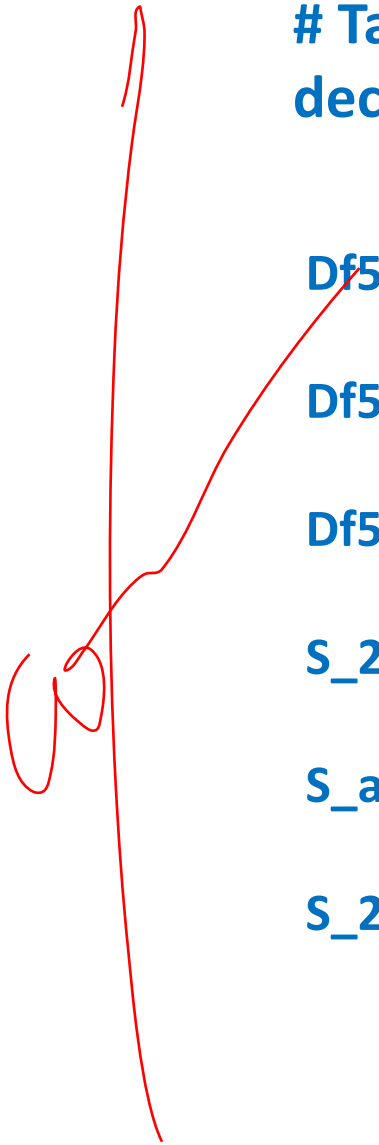
 # Task 5: Find out what fraction of planets have been found in the last decase(2010s) across each method type

 # Step 1: Filter the data for given condition (planets found in last decade)
 # Step 2: Split based on the method
 # Step 3: Apply count function
 # Step 4: Aggregate the final results



PRIME INTUIT
Finishing School

Pandas – Data Frame Objects



 # Task 5: Find out what fraction of planets have been found in the last decase(2010s) across each method type

```
Df5[df5.year >= 2010]

Df5[df5.year >= 2010].groupby('method')

Df5[df5.year >= 2010].groupby('method')['method'].count()

S_2010s = Df5[df5.year >= 2010].groupby('method')['method'].count()

S_alltime = Df5.groupby('method')['method'].count()

S_2010s / s_alltime
```