Pandas - DA

Tuesday 18 March 2025 09:07

Data Analytics :

```
df = pd.read_csv('df')
```

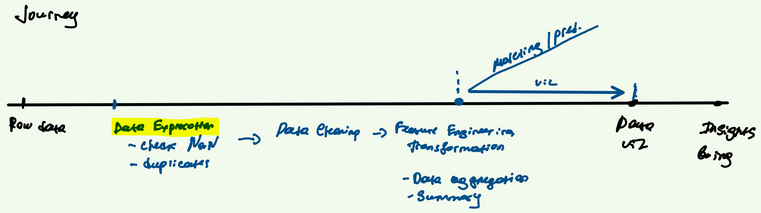
- Data Exploration

```
df.shape / columns
df.describe()
df.info
```
} General

```
df.isna().values.any()
df.duplicated().values.any()
```
} Missing / Duplicates

```
df.isna().sum()
df.duplicated().sum()
```
—> `duplicates = df[df.duplicated()]` to see duplicates/missing

Journey



keep it

Droping

```
clean = df.dropna() / df.dropna(inplace=True)
clean = df.dropna(subset=['column_1', 'column_2'])
```
dropping in specific columns

```
clean = df.drop_duplicates() / df.drop_duplicates(inplace=True)
clean = df.drop_duplicates(subset=['column_1', 'column_2'])
df = df.drop(column='...', axis=1)
```

Filling NaN values + Median, Mean, Mode

```
df.fillna(0, inplace=True) / df['column'] = df['column'].fillna(0)
```
0 to be replaced with

```
df['numeric_col'] = df['numeric_col'].fillna(df['numeric_col'].mean())
```
Median() mode() ← categorical data

Forward Fill / Backward Fill ← replacing with back/forward value

```
df.fillna(method='ffill', inplace=True)
'bfill'
```

Predicting

```
imputer = KNNImputer(n_neighbors=3)
df_imputed = pd.DataFrame(imputer.fit_transform(df), columns=df.columns)
```

Converting data

```
df['date'] = pd.to_datetime(df['date']) —> df['year'] = df['date'].dt.year/month/day
df['num'] = pd.to_numeric(df['num']) —> handling error : my data [1, 2, three, 4]
                                                                  to NaN
df['num'], errors='coerce')
df['category'] = df['category'].astype('category')
df['Boolean'] = df['Boolean'].map({'true': True, 'False': False})
df['price'] = df['price'].replace('[\$,]', '', regex=True).astype(float)
```

Loc & iloc

Name — loc
int position — iloc

```
df.loc['row1': 'row3']
filtering = df.loc[df['column'] > threshold]
df.iloc[0,0] - First row, First column
df.iloc[0:3] - First 3 rows
df.iloc[:,1] - Second column
df.iloc[1:4, 0:2] - specific rows, specific columns
```

Sorting & Grouping — Summarizing

```
df.sort_values("column", ascending=True)
df.value_counts() ← Grouping and counting only unique data
df.group_by('group')
df.group_by('group').size() ← Counting total + NaN
                .count() ← Counts all existing values
```
I Simple grouping => more for me

```
agg() - Summarizing + aggregating.
df.agg('mean') ...   for each column
df.groupby('...').agg({'<column>': 'count'})
```

```
df.pivot_table(values='<column>', index='...', aggfunc='...',
               aggfunc='mean')
```
=> Better for Deep summary — Reshaping

Outliers

- Take a look with Box plot

```
sns.boxplot(x=df['values'])
plt.title("Largest outliers")
plt.show()
```

- Take a look with histogram

```
df['values'].hist(bins=30)
plt.title(...)
plt.show()
```

Handling with Stats !

```
Q1 = df['values'].quantile(0.25)
Q3 = df['values'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
Outliers = df[(df['values'] < lower_bound) | (df['values'] > upper_bound)]
```
↳ Remove or KEEP ← called Cap or Winsorize

Feature Engineering

- Creating new columns

```
df['new'] = df['col'] / df['col_2']
```

- Feature Extraction

```
df['year'] = df['date'].dt.year
```

- Feature transformation
  Not for now

Merging

```
merged_df = pd.merge(df_1, df_2, on='id')
```

Querying

```
df.query("'column_1' > 2") - From this column_1 -> Show only values bigger than 2
ex_values = 5
df.query("'column_1' > @ex_values")
```
OR `df = df[df['column_1'] < 5]`

Data Viz

MatPlotLib

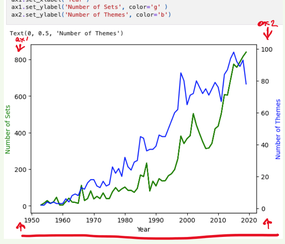
```
ax1 = plt.gca()
ax2 = ax1.twinx()
```
} 2 plots with dif. axes

Customization :

- linestyle = '-', '--', ':', '-.'
- Color = 'blue' / column
- marker = 'o'
- size — `plt.figure(figsize=(14,8), dpi=120)`

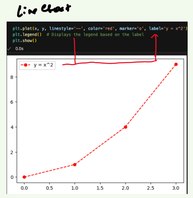
Line Charts with Two Seperate Axes

```
ax1.plot(sets_by_year[2:-2], color='g')
ax2.plot(themes_by_year[2:-2], color='b')

ax1.set_xlabel('Year')
ax1.set_ylabel('Number of Sets', color='g')
ax2.set_ylabel('Number of Themes', color='b')
```



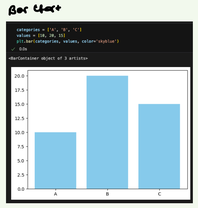
Line chart

```
plt.plot(x, y, linestyle='--', color='red', marker='o', label='y = x^2')
plt.legend() # Displays the legend based on the label
plt.show()
```

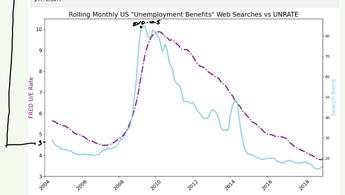


Bar chart

```
categories = ['A', 'B', 'C']
values = [10, 20, 15]
plt.bar(categories, values, color='skyblue')
```

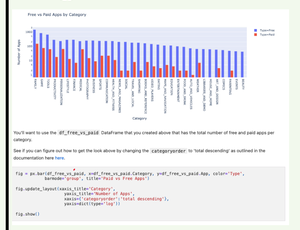


```
plt.figure(figsize=(14,8), dpi=120) # size
plt.title('Rolling Monthly US "Unemployment Benefits" Web Searches vs UNRATE', fontsize=18)
plt.yticks(fontsize=14)
plt.xticks(fontsize=14, rotation=45)

ax1 = plt.gca()
ax2 = ax1.twinx()

# ax1.xaxis.set_major_locator(years)
# ax1.xaxis.set_major_formatter(years_fmt)
# ax1.xaxis.set_minor_locator(months)

ax1.set_ylabel('FRED U/E Rate', color='purple', fontsize=16)
ax2.set_ylabel('Search Trend', color='skyblue', fontsize=16)

ax1.set_xlim(bottom=3, top=18.5) # Setting the axes to end
ax1.set_xlim([df_unemployment.MONTH[0], df_unemployment.MONTH.max()])

# Calculate the rolling average over a 6 month window
roll_df = df_unemployment[['UE_BENEFITS_WEB_SEARCH', 'UNRATE']].rolling(window=6).mean()

ax1.plot(df_unemployment.MONTH, roll_df.UNRATE, 'purple', linewidth=3, linestyle='--')
ax2.plot(df_unemployment.MONTH, roll_df.UE_BENEFITS_WEB_SEARCH, 'skyblue', linewidth=3)

plt.show()
```
=> smoothing



Structure pretty much same

Plotly - Dynamic



You'll want to use the `df_free_vs_paid` DataFrame that you created above that has the total number of free and paid apps per category.

See if you can figure out how to get the look above by changing the `categoryorder` to 'total descending' as outlined in the documentation here here.

```
fig = px.bar(df_free_vs_paid, x=df_free_vs_paid.Category, y=df_free_vs_paid.App, color='Type',
             barmode='group', title='Paid vs Free Apps')

fig.update_layout(xaxis_title='Category',
                  yaxis_title='Number of Apps',
                  xaxis={'categoryorder':'total descending'},
                  yaxis=dict(type='log'))

fig.show()
```

SEABORN

```
plt.figure(figsize=(8,4), dpi=200)
with sns.axes_style("darkgrid"):
  ax = sns.regplot(data=new_films,
                   x="Budget in $100 millions",
                   y="US0_Worldwide_Gross",
                   color="#2ff8d7",
                   scatter_kws = {'alpha': 0.3},
                   line_kws = {'color': '#ff7c43'})

ax.set(ylim=(0, 3000000000),
       xlim=(0, 4500000000),
       ylabel='Revenue in $ billions',
       xlabel='Budget in $100 millions')

[(0.0, 3000000000.0),
 (0.0, 4500000000.0),
 Text(0, 0.5, 'Revenue in $ billions'),
 Text(0.5, 0, 'Budget in $100 millions')]
```