Date: 30-04-2025

1 Test 1 - EDA
2 Test 2 - Stats

To do:

1b Stats sumup (pay-2)
2 EDA sumup
3 Stats imp. python snippets
4a EDA imp. python snippets
5 Case study in class & takehome
6 Interview Qns - Stats.
7 Interview Qns - EDA
8b Stats other concepts (pay-1,2)
9a Replace remove binning date func drop Drill drill string func

EDA Sum up

Flow of process

1 Read the data
↓
2 Basic understanding of data
↓
3 Find duplicated rows & remove if any
↓
4 Find anomalies and address them (for each column)
(e.g. spelling mistake, wrongly identified datatype, Improper values - ages as -ve, Salary as -ve, etc)
↓
5 Feature Engineering
(Create new feature from existing and remove existing columns)
e.g. BMI, binning (for numeric column) vv. imp.

Python snippets for each

1 pd.read_csv( )
2 df.info(), df.shape, df.size
[Note: pandas is on top of numpy]
↳ so attributes & methods.
3 a df.duplicated( ).sum()
b df[~df.duplicated()]
4 *h df[col].str.replace(old, new, regex)
a df[col].replace(dict) → old (key), new (value)
b df[col].astype('int')
c df[col].astype(str)
d df[col].astype('category')
e pd.to_numeric(df[col], errors='coerce')
*f pd.to_datetime(df[col], errors='coerce')
g df[col].astype(float)
5 a pd.cut(df[col], bins=, → interval, labels=, → name intervals, include_lowest=True)
b df['salary'].transform(lambda x: x/x.max())
c df['salary'].apply(lambda x: x*1.1)
df['dept'].map(dict) → old: key, new: value

Extra:
* df['age'].apply(lambda x: abs(x) if x<0 else x)
* df.groupby('dept')['age'].transform(lambda x: x.fillna(x.median()))
* df.dropna() → drop rows with any missing value
* df.drop('colname', axis=1) → specific column drop
* df.dropna(subset=[col name])

6 Select numeric and categoric columns seperately

a. df.select_dtypes(include=np.number).columns.to_list()

b. df.select_dtypes(exclude=np.number).columns.to_list()

7 Basic descriptive stastistics (five point summary)

a. df[num].describe()

b. df[cat].describe()

8 Treat missing values

a. df['age'].fillna(df['age'].median())

b. df[col].bfill()

c. df[col].ffill()

d. df['age'].apply(lambda x: df['age'].median() if x<0 else x).fillna(df.groupby('dept')['age'].transform('median'))

<10% ⓐ Single value treatment - mean, median [Num] mode [cat]

ⓑ Multi value treatment - bfill, ffill

>10% & <40% ⓒ logical imputation

ⓓ remove the observation / row / column <5% >40%

9 Treat "outliers" [only for numeric columns] [as cat. only imbalance] [entire row is outlier if only one value in it is outlier]

Identify: ⓐ IQR ⓑ Z-score

Treat: ⓐ Remove (<5%) ⓑ Capping with upper / lower whicker ⓒ Transformation (yeojohnson, Boxcox, etc)

list of num. col. names

Q_1 = df[num].quantile(0.25)

Q_3 = df[num].quantile(0.75)

iqr = $Q_3 - Q_1$

uw = $Q_3 + 1.5 * iqr$

lw = $Q_1 - 1.5 * iqr$

df[(df[num] < lw) | (df[num] > uw)).any(axis=1)]

↳ to identify the outliers (rows)

10 Advance descriptive stastistics [To get claims for sample]

ⓐ univariant analysis
ⓑ Bivariant analysis
ⓒ multivariant analysis
} for each metric & viz.

11 Inferential Stastistics [mainly [hypo. testing] & estimation]

ⓐ one sample testing - z, t test
ⓑ Two sample testing - z, t test, mannwhitenegu
ⓒ more than two sample testing - Anova, Tukeyhsd, Kruskal wallis
ⓓ Categorical columns - chisquare
ⓔ pearson r

Feature Engineering (for Model)

ⓐ For categorical columns

Encoding

ⓑ For numerical columns

1st Transformation (if needed)

↓

2nd Scaling (if needed)

Advance Descriptive Statistics

Need: 1 Advance descrip. (basic) 2 Inferential stats 3 FE (for Model)

(remember by diagram o/p for arguments)

Univariant analysis

Metric		Python code
mean	Numeric	df[col].mean()
median		df[col].median()
mode		df[col].mode()[0]
IQR		df[col].quantile(0.75) - df[col].quantile(0.25)
Skewness		df[col].skew()
kurtosis		df[col].kurt()
mode	Categoric	df[col].mode()[0]
value_counts		df[col].value_counts() or

Plot		Python code
Histogram	Numeric	sns.histplot(df[num])
kde		sns.kdeplot(df[num])
dist (Hist + kde)		sns.distplot(df[num])
boxplot		sns.boxplot(x=df[num])
Countplot	Categoric	sns.countplot(x=df[cat], palette='magma')

Piechart

```
count = df[cat].value_counts()
count.plot(kind='pie', labels=count.index
           , autopct='%.%.2f%%')
```

Bivariant analysis

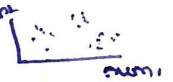
Metric / plot

Num vs Num

- Correlation

```
df[num1].corr(df[num2])
```

- Scatterplot




```
sns.scatterplot(x=df[num1],
                y=df[num2])
```

- lineplot


```
plt.plot(df[num1], df[num2], '*g',
         linestyle='--')
```

Num vs Cat

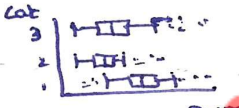
- Groupby

```
df.groupby('cat')[num].mean()
```

- kdeplot


```
sns.kdeplot(x=df[num], hue=df[cat])
```

- boxplot

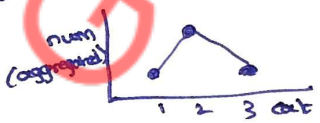



```
sns.boxplot(x=df[num], y=df[cat])
```

- barplot


```
sns.barplot(x=df[cat], y=df[num])
```

- lineplot



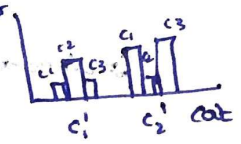

```
sns.lineplot(x=df[cat], y=df[num])
```

Cat vs Cat

- Cross tab

```
pd.crosstab(df[cat1], df[cat2])
```

- Countplot



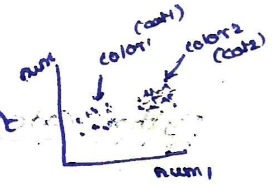

```
sns.countplot(x=df[cat1], hue=
              df[cat2])
```

Multivariant analysis

plot

Num vs Num vs Cat

Scatter plot

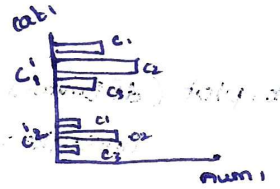


Python code

```
Sns. Scatterplot (df[num1], y=df[num2],
                  hue=df[cat])
```

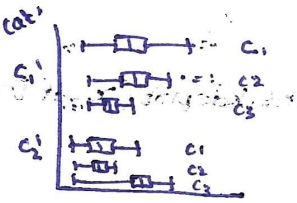
Cat vs Cat vs Num

Boxplot




```
Sns. boxplot (x=df[num1], y = df[cat1],
              hue=df[cat2])
```

Boxplot

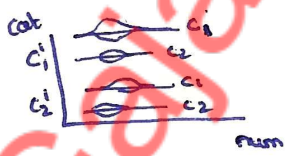



```
Sns.boxplot (x = df[num1], y = df [cat1],
             hue=df[cat2])
```

Extra plot

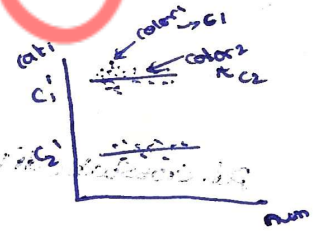
Cat vs Cat vs Num

Violin plot




```
Sns.violin plot (x=df[num1], y=df[cat1],
                 hue=df[cat2])
```

Swarm plot




```
Sns.swarmplot (x = df[num1], y =df
               [cat1], hue =df[cat2])
```

Heatmap

a) Correlation

b) nullvalues

```
Sns.heatmap (df.corr(), cmap='Green',
             annot=True)
Sns.heatmap (df.isnull(), cbar = False)
```

B/w numeric values ← Pairplot

a) with hue = categoric

b) w/o hue = categoric

```
Sns.pairplot (df, hue=df[cat1])
Sns.pairplot (df)
```

Inferential Statistics

Num vs Num

① Pearson r

Prerequisite:
a. Data normal
b. Equal variance (so different units) as num vs num

H_0: not related variables/samples

H_1: Related Sample

python code

Stats.pearsonr(Sam1, Sam2)

Num vs Cat

① If Cat = 2 (Sample)

ⓐ Two Sample T-test

	H_0	H_1
	=	!=
	≤	>
	≥	<

[on μ vs μ0]

Pre-requisite:
a. Independent Samples
b. Data normal
c. equal variance or not

↓ if not met then Mannwhitneyu

① If equal variance

Stats.ttest_ind(Sam1, Sam2, alternative=...)

② If unequal variance

Stats.ttest_ind(Sam1, Sam2, alternative=..., equal_var=False)

② If Cat > 2 (Samples)

ⓐ ANOVA

Prerequisite:
a. Independent Samples
b. Data normal
c. equal variance

↓ if not met then Kruskal wallis.

H_0: all μ are same

H_1: atleast one μ is different

Stats.f_oneway(Sam1, Sam2, ----)

ⓑ Tukey hsd (after anova) ↳ to find exactly

H_0: =

H_1: ≠

```
import statsmodels.stats.multicomp as mc
report = mc.pairwise_tukeyhsd(df[num], df[cat])
report.summary()
```

Cat vs Cat

① Chi square contingency

H_0: not related

H_1: realted

Stats.chi2_contingency(observed data) ↳ is crosstab

finally:

1. Reject Ho
2. fail to reject Ho

Feature Engineering

Encoding

Ordinal Encoding

One Hot Encoding

(N-1) dummies

Freq. encoding

Label Encoding

Scaling / Data normalization

Standar Scaler (Z Score)

$\mu = 0, \sigma = 1$

Min-max

[0, 1]

Python code

```
from sklearn.preprocessing import OrdinalEncoder
O = OrdinalEncoder(category=['low', 'med', 'High'])
df[[col]] = O.fit_transform(df[[col]])
```

```
df_new = pd.get_dummies(df, columns=
        [col1, col2, ...], dtype=int/float)
```

```
df_new = pd.get_dummies(df, columns=[...],
        dtype=int/float, drop_first=True)
```

```
freq = df[col].value_counts(normalize=True)
df[col] = df[col].replace(freq)
```

```
from sklearn.preprocessing import LabelEncoder
l = LabelEncoder()
df[[col]] = l.fit_transform(df[[col]])
```

```
from sklearn.preprocessing import StandardScaler
S = StandardScaler()
df[col] = S.fit_transform(df[[col]])
```

```
from sklearn.preprocessing import MinMaxScaler
m = MinMaxScaler()
df[col] = m.fit_transform(df[[col]])
```

Robust Scaler

$\frac{x - \text{median}}{IQR}$

```
from sklearn.preprocessing import RobustScaler
r = RobustScaler()
df[col] = r.fit_transform(df[[col]])
```

Data Transformation

Box-cox (>0)

```
from sklearn.preprocessing import PowerTransformer
p = PowerTransformer(method = 'box-cox')
df[col] = p.fit_transform(df[[col]])
```

Yeo-johnson (+,-,0)

```
from sklearn.preprocessing import PowerTransformer
p = PowerTransformer()
df[col] = p.fit_transform(df[[col]])
```

Date handling

* pd.to_datetime(.)
  - expected i/p format: %m/%d/%Y %H:%M
  - format = 'mixed' => if 01/11 as months in diff. rows. (Aug.)
  - errors = 'coerce'
    - if i/p: Null, String, then => o/p: NAT (Not available time stamp)

* date & time extract
  - df[col].dt.date
  - df[col].dt.time

String handling

e.g.: 1 df[col].str.upper()

2 df[col].str[:3]

3 df[col].str.contains('Tech', case=False, na=False)

4 df[col].str.replace('Human', 'people', case = False)