[A/B Testing and Experimentation with Pandas] #cheatsheet

1. Data Preparation for A/B Testing

- **Loading Experiment Data**: `df = pd.read_csv('experiment_data.csv')`
- **Splitting Test Groups**: `group_a = df[df['group'] == 'A'], group_b = df[df['group'] == 'B']`
- **Ensuring Equal Distribution**: `df['group'].value_counts()`
- **Setting Date as Index (if time series)**: `df['date'] = pd.to_datetime(df['date']), df.set_index('date', inplace=True)`

2. Initial Data Exploration

- **Summary Statistics per Group**: `df.groupby('group').describe()`
- **Checking for Missing Values**: `df.isnull().sum()`
- **Visualizing Distribution of Key Metrics**: `df.groupby('group')['metric'].plot(kind='hist', alpha=0.5)`

3. Metric Calculation

- **Calculating Conversion Rates**: `conversion_rate = df.groupby('group')['conversion'].mean()`
- **Calculating Average Order Value**: `avg_order_value = df.groupby('group')['order_value'].mean()`
- **Calculating Click-Through Rate (CTR)**: `ctr = df.groupby('group')['click'].mean()`

4. Testing for Normality

- **Shapiro-Wilk Test**: `from scipy.stats import shapiro; shapiro(group_a['metric'])`
- **Visual Check with Histograms**: `group_a['metric'].hist()`

5. Testing for Variance Homogeneity

- **Levene's Test for Equal Variances**: `from scipy.stats import levene; levene(group_a['metric'], group_b['metric'])`

By: Waleed Mousa

6. Statistical Testing

- **T-Test for Mean Comparison**: `from scipy.stats import ttest_ind; ttest_ind(group_a['metric'], group_b['metric'])`
- **Mann-Whitney U Test for Non-Parametric Data**: `from scipy.stats import mannwhitneyu; mannwhitneyu(group_a['metric'], group_b['metric'])`

7. Effect Size Calculation

- **Calculating Cohen's d**: `def cohens_d(x, y): ...; cohens_d(group_a['metric'], group_b['metric'])`

8. Confidence Intervals

- **Bootstrap Confidence Intervals**: `def bootstrap_ci(df, metric, n=1000): ...; bootstrap_ci(df, 'metric')`

9. Power and Sample Size Analysis

- **Calculating Power of the Test**: `from statsmodels.stats.power import TTestIndPower; power = TTestIndPower().solve_power(...)`
- **Sample Size Determination**: `from statsmodels.stats.power import TTestIndPower; sample_size = TTestIndPower().solve_power(...)`

10. Visualizing Results

- **Boxplots for Metric Comparison**: `df.boxplot(by='group', column=['metric'])`
- **Bar Chart for Conversion Rates**: `conversion_rate.plot(kind='bar')`

11. Handling Time Series Data

- **Cumulative Metrics Over Time**: `df.groupby(['date', 'group']).agg({'metric':'cumsum'})`
- **Time Series Plot for Cumulative Metrics**: `df.groupby(['date', 'group']).agg({'metric':'cumsum'}).unstack().plot()`
- **Analyzing Daily Trends**: `df.groupby([df.index.day, 'group']).mean()`

- **Weekday vs. Weekend Analysis**: `df['weekday'] = df.index.weekday; df.groupby(['weekday', 'group']).mean()`

12. Segmentation Analysis

- **Segmented Metrics Analysis**: `df.groupby(['segment', 'group']).agg({'metric': 'mean'})`
- **Segmented Statistical Testing**: `segmented_ttest = df.groupby('segment').apply(lambda x: ttest_ind(x[x['group'] == 'A']['metric'], x[x['group'] == 'B']['metric']))`

13. Regression Analysis for A/B Testing

- **Logistic Regression for Conversion**: `from statsmodels.api import Logit; Logit(df['converted'], df[['intercept', 'group']]).fit()`
- **Linear Regression for Continuous Outcomes**: `from statsmodels.api import OLS; OLS(df['metric'], df[['intercept', 'group']]).fit()`

14. Bayesian Approaches

- **Bayesian A/B Testing**: `import pymc3 as pm; with pm.Model() as model: ...; pm.sample(...)`
- **Posterior Probability Distributions**: `pm.plot_posterior(...)`

15. Handling Multiple Comparisons

- **Bonferroni Correction**: `from statsmodels.sandbox.stats.multicomp import multipletests; p_adjusted = multipletests(pvals, method='bonferroni')`

16. Data Transformation and Feature Engineering

- **Creating Interaction Features**: `df['interaction_feature'] = df['feature1'] * df['feature2']`
- **Encoding Categorical Variables**: `df = pd.get_dummies(df, columns=['categorical_feature'])`
- **Normalizing Continuous Variables**: `df['normalized_feature'] = (df['feature'] - df['feature'].mean()) / df['feature'].std()`

17. User Behavior Analysis

- **Session Duration Analysis**: `session_duration = df.groupby(['user_id', 'group'])['duration'].sum()`
- **Frequency of User Actions**: `action_frequency = df.groupby(['user_id', 'action']).count()`

18. Cohort Analysis

- **Creating User Cohorts**: `df['cohort'] = df.groupby('user_id')['date'].transform('min')`
- **Cohort Retention Analysis**: `cohort_retention = df.pivot_table(index='cohort', columns='group', values='retention_rate')`

19. Dealing with Outliers

- **Identifying Outliers**: `df['metric'].quantile([0.01, 0.99])`
- **Winsorizing Data**: `from scipy.stats.mstats import winsorize; df['winsorized_metric'] = winsorize(df['metric'], limits=[0.01, 0.01])`

20. Advanced Visualization

- **Cohort Analysis Heatmap**: `sns.heatmap(cohort_retention, annot=True)`
- **Cumulative Conversion Rate Over Time**: `df.groupby(['date', 'group'])['conversion'].cumsum().unstack().plot()`

21. Experiment Duration Analysis

- **Minimum Detectable Effect Calculation**: `from statsmodels.stats.power import tt_ind_solve_power; tt_ind_solve_power(effect_size=..., alpha=..., power=...)`
- **Running Time Calculation**: `running_time = df['date'].max() - df['date'].min()`

22. Multi-Variant Testing

- **Analyzing Multi-Variant Tests**: `multivariant_df.groupby(['variant', 'group']).agg({'metric': 'mean'})`

- **Statistical Testing for Multi-Variant**: `f_oneway(multivariant_df[multivariant_df['variant'] == 'A']['metric'], multivariant_df[multivariant_df['variant'] == 'B']['metric'], ...)`

23. Post-Experiment Analysis

- **Long-Term Impact Analysis**: `long_term_df.groupby('group')['long_term_metric'].mean()`
- **User Feedback and Qualitative Analysis**: Incorporating qualitative data and user feedback post-experiment.

24. Data Quality Checks

- **Checking for Consistency in Groups**: `df.groupby('group').apply(lambda x: x['metric'].std() / x['metric'].mean())`
- **Data Completeness Check**: `df.groupby('group').count()`

25. Advanced Statistical Techniques

- **Propensity Score Matching**: Using logistic regression to match users in different groups based on propensity scores.
- **Survival Analysis for Duration Metrics**: `from lifelines import KaplanMeierFitter; kmf = KaplanMeierFitter(); kmf.fit(durations=df['duration'], event_observed=df['event'])`