1. Basic Python & Pandas Overview

- Basic Python can sum, mode without libraries.
- Pandas enables efficient file reading and data manipulation.
- Libraries enhance speed and precision; list for collections.
- Magic commands: %magic for commands list, %timeit for timing code, %whos for active variables.

2. Data Handling in Pandas

Loading & Cleaning:
- np.array, pd.Series for creating arrays/series.
- pd.DataFrame for creating and summarizing data tables.
- Indexing: .iloc[] for position, .loc[] for label.
- Sorting: .sort_values(), .sort_index().
- Filtering: Boolean indexing e.g., data[data['Age'] > 40].
- Statistics: .mean(), .unique(), .value_counts().

3. Data Manipulation

- Importing Data:
  - Use pd.read_csv, pd.read_json, pd.read_html with parameters for formatting.
- Cleaning Data:
  - Replace or convert data types, handle missing values with .dropna().
- Categorizing & Joining:
  - Create dummies, concatenate, and join data frames.
- Reshaping Data:
  - Index manipulation: Set/reset index, hierarchical indexing.

stack()/unstack() for pivoting data.
- Combining Data:
  - join(), concat(), merge() for combining datasets.
- Pivoting:
  - pivot() for reshaping data without aggregation.
  - pivot_table() for reshaping with aggregation.

4. Data Visualization

Matplotlib.pyplot:
- Plotting:
  - Line plots: plt.plot(t, t**2, marker, linestyle, color).
  - Other types: plt.scatter(), plt.bar(), plt.hist().
- Figure and Axes:
  - Create figures: plt.figure().
  - Add axes: plt.axes(), plt.subplot().
- Customization:
  - Titles, labels: plt.title(), plt.xlabel(), plt.ylabel().
  - Axis ticks: plt.xticks(), plt.yticks().
  - Display plot: plt.show().
  - Customize lines: plt.setp(line, properties).
- Annotations:
  - Adding text: plt.text(position, text).
  - Use for mean, SD, etc.
- Grouped Bar Plots:
  - Using groupby() and unstack().
  - Example: Stacked bar chart for 'Sugary' vs. 'Non-Sugary' items.

Seaborn:
- Distribution Plots:
  - Swarm plot: sns.swarmplot(x), shows value distribution with jitter.
  - Violin plot: sns.violinplot(x), combines box plot with kernel density estimate.
  - Box plot: sns.boxplot(x), shows min, max, quartiles, outliers.
  - Histogram: sns.histplot(data), for univariate distribution.
- Relational Plots:
  - Relational plot: sns.relplot(x, y, hue, size, alpha, data).
  - Example: Relationship between two variables, colored by category.
- Pairplot:
  - Breakfast: sns.pairplot(menu.query("Category == 'Breakfast'"), vars=['Calories', 'Total Fat', 'Protein', 'Dietary Fiber']).
  - General: sns.pairplot(data=menu, hue="Category", vars=['Calories', 'Total Fat', 'Protein', 'Dietary Fiber']).
- QQ Plot:
  - Quantile comparison: sns.regplot(y=xr, x=qntls).
- Histogram & Skewness:
  - Skewness: sns.histplot(leftskewed, kde=False).
  - QQ plot: sns.regplot(x=xr, y=qntls).

5. Regression Analysis & Visualization

Considering example from class
- Correlation Analysis:
  - Calculated wine dataset correlations.: .corr()
  - Filtered and sorted unique pairs by absolute values.
- Joint Plot Visualization:
  - Created with sns.JointGrid for "free sulfur dioxide" vs. "total sulfur dioxide".
  - Plotted regression and histograms on the margins.
- OLS Regression Results:
  - Model1: Dependent - "total sulfur dioxide"; $R^2 = 0.446$.
  - Model2: Dependent - "pH"; $R^2 = 0.117$.
  - Coefficients, standard errors, t-values, and p-values are provided.
- Statistical Significance:
  - F-statistics and Prob(F-statistics) indicate model significance.
- Estimations:
  - Predicted "total sulfur dioxide" from "free sulfur dioxide".
  - Forecasted "apparent temperature" from "temperature" and "humidity".

6. . Statistical Analysis Summary

- ANOVA & T-tests:
  - model = ols('log_box_office_dollars ~ C(person_of_color)', data).fit()
  - ANOVA p-value: 0.0428; T-test p-value: 0.0428.
- Regression Insight:
  - R^2: 0.009; weak effect.
  - Skewness: -0.745; data not symmetrical.
- Data Parsing:
  - Parse earnings: re.match(r'\$([0-9.]+)([KM]?)', earnings)
  - Convert strings: "M" → *1e6, "K" → *1e3.
- Model Validity:
  - Consistent p-values indicate significant differences.
  - Skew and Omnibus test for normality.

7. Contingency Analysis

- Setup:
  - Crosstab & pivot for contingency: pd.crosstab(df.color, df.vehicleClass).
- Visualization
  - Heatmap: sns.heatmap(ct, annot=False).
  - Mosaic plot: mosaic(data, ['passtype', 'status']).
- Expected Values:
  - Calculate per cell: row_total * col_total / overall_total.
- Chi-Square Test:
  - Test for independence: chi2_contingency(ct).
  - Null hypothesis: No difference in groups.

8. **Text Processing with Pandas & Regex**

- Case Conversion:
  - To lower: .str.lower() - To upper: .str.upper()
- Length Calculation:
  - String length: .str.len()
- String Operations:
  - Split: .str.split('00').str.get(1)
  - Replace: .str.replace('dog', 'health', regex=True)
- Regex Extraction:
  - Single match: .str.extract(r'(Dog)')
  - Multiple matches: .str.extract(r'(Dog|Taffy)')
  - Case-insensitive: .str.extract(r'(Dog|[Tt]affy)', expand=False)
  - All matches: .str.extractall(r'(Dog|[Tt]affy)')

9. **NLP Fundamentals & spaCy Overview**

- Initialize spaCy: nlp = spacy.load('en_core_web_sm').
- Text manipulation: .lower(), .upper(), .len(), .split(), .replace().
- Tokenization: Convert text into tokens (nlp("Hello World!")).
- Sentence detection: doc.sents for sentence tokens.
- Named Entity Recognition (NER): [(X.text, X.label_) for X in doc.ents].
- Regex for text cleaning: re.match(r'\$([0-9.]+)([KM]?)', earnings).
- Word embedding with gensim: Load Word2Vec model.
- Remove punctuation/special chars: .translate(str.maketrans('', '', string.punctuation)).
- Count word frequency: defaultdict(int) and sns.countplot.
- Remove stop words: Filter out STOP_WORDS.
- Calculate word similarity: `w2v_mod['word'].shape`

10. Machine Learning with scikit-learn:

- Linear Regression Workflow:
  - Import and initialize: `from sklearn.linear_model import LinearRegression`
  - Instantiate model: `lm = LinearRegression()`
  - Prepare data: `X = df['feature'], y = df['target']`
  - Fit model: `lm.fit(X, y)`
  - Predict: `lm.predict([[new_value]])`
  - Model evaluation: `lm.score(X, y),lm.coef_,lm.intercept_`
- Cross-Validation:
  - Import: `from sklearn.model_selection import cross_validate`
  - Execute: `results = cross_validate(lm, X, y, scoring='neg_mean_squared_error')`
  - Test scores: `results['test_score']`
- Data Preprocessing:
  - Missing value check: `df.isnull().sum()`
  - Remove missing values: `df.dropna(inplace=True)`
  - Create train-test split: `train_test_split(X, y, stratify=y)`
- Pipeline:
  - Import: `from sklearn.pipeline import Pipeline`
  - Create pipeline: `Pipeline(steps=[('scaling', StandardScaler()), ('encode', OneHotEncoder())])`
  - Fit and transform: `pipeline.fit_transform(X)`
- Visualization:
  - Distribution plot: `sns.displot(data=df, x='feature', hue='category')`
  - Pairplot: `sns.pairplot(data=df, hue='category')`

13. Classification

- Metrics:
  - Accuracy: Correct predictions ratio.
  - Precision: True positives over all positives.
  - Recall: True positives over actual positives.
  - F1 Score: Harmonic mean of Precision and Recall.
- Models:
  - SVM: RBF kernel, accuracy evaluation.
  - Decision Tree: Entropy-based, max depth control.
  - Random Forest: Ensemble of decision trees, cross-validation for accuracy.
- Code Snippets:
  - `svm.SVC().fit(X_train, y_train)`
  - `DecisionTreeClassifier().fit(X_train, y_train)`
  - `RandomForestClassifier().fit(X_train, y_train)`
- Tuning:
  - Adjust `n_estimators`, `max_depth`.
  - Use `GridSearchCV` for best parameters.

11. Dimensionality Reduction & Visualization Techniques

- PCA (Principal Component Analysis):
  - Standardize data: `scale(X)`
  - Reduce dimensions: `PCA(n_components=2)`
  - Transform data: `X_pca = pca.fit_transform(X)`
  - Explained variance: `pca.explained_variance_ratio_`
- MDS (Multi-dimensional Scaling):
  - Configure MDS: `manifold.MDS(n_components=2, metric=False)`
  - Fit model: `nmds.fit_transform(X)`
- t-SNE (t-distributed Stochastic Neighbor Embedding):
  - Setup t-SNE: `TSNE(n_components=2, perplexity=40)`
  - Apply t-SNE: `X_tsne = tsne.fit_transform(X_norm)`
- Image Compression with PCA:
  - Decompose image channels: `PCA(n_components=50)`
  - Reconstruct images: `pca.inverse_transform(channel_transformed)`
- Plotting:
  - PCA scatter: `px.scatter(x=X_pca[:, 0], y=X_pca[:, 1], color=y)`
  - t-SNE scatter: `px.scatter(x=X_tsne[:, 0], y=X_tsne[:, 1], color=y)`
- scikit-learn Pipelines:
  - Read data: `pd.read_csv(filepath)`
  - Scale and PCA: `Pipeline(steps=[('scaling', StandardScaler()), ('pca', PCA(n_components=2))])`
  - Fit and predict: `pipeline.fit(X); pipeline.predict(X)`
- Cross-Validation:
  - Validate model: `cross_validate(model, X, y, scoring='score')`
- Churn Analysis with PCA:
  - Visualize customer churn:
  - PCA transformed data in scatter plot.

12. Clustering:

- Preprocess Data:
  - Normalize with `StandardScaler`.
  - Reduce dimensions via `PCA`.
- Clustering:
  - Apply `KMeans` and `AgglomerativeClustering`.
  - Assess with silhouette scores.
- Pipeline:
  - Combine preprocessing and clustering steps in `Pipeline`.
- Visualize:
  - Dendrogram for hierarchical clusters.
  - PCA scatter plot for KMeans clusters.
- Code Snippets:

```
# Standardize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# PCA for dimension reduction
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# KMeans Clustering
kmeans = KMeans(n_clusters=3)
labels = kmeans.fit_predict(X_pca)
```

- Parameter Tuning:
  - Adjust `n_components` in PCA.
  - Modify `n_clusters` for KMeans.
- Evaluation:
  - Calculate explained variance in PCA.
  - Use silhouette score for KMeans optimization.

13. DASK

- dask.dataframe can scale to much larger datasets.
- implements a blocked parallel DataFrame object that mimics a significant subset of the Pandas DataFrame API
- Dask DataFrames parallelize computations across partitions, akin to Pandas but optimized for efficiency