[Feature Engineering] (CheatSheet)

1. Basic Feature Creation

- **Combining Features**: `df['new_feature'] = df['feature1'] + df['feature2']`
- **Differences Between Features**: `df['new_feature'] = df['feature1'] - df['feature2']`
- **Ratios of Features**: `df['new_feature'] = df['feature1'] / df['feature2']`
- **Product of Features**: `df['new_feature'] = df['feature1'] * df['feature2']`
- **Log Transformation**: `df['log_feature'] = np.log(df['feature'] + 1)`

2. Handling Categorical Features

- **One-Hot Encoding**: `pd.get_dummies(df['categorical_feature'])`
- **Label Encoding**: `sklearn.preprocessing.LabelEncoder().fit_transform(df['categorical_feature'])`
- **Binary Encoding**: `category_encoders.BinaryEncoder().fit_transform(df['categorical_feature'])`
- **Frequency Encoding**: `df.groupby('category').size() / len(df)`
- **Mean Encoding**: `df.groupby('category')['target'].mean()`

3. Temporal Features

- **Extracting Day, Month, Year**: `df['day'] = df['datetime'].dt.day`
- **Extracting Weekday**: `df['weekday'] = df['datetime'].dt.weekday`
- **Extracting Hour, Minute, Second**: `df['hour'] = df['datetime'].dt.hour`
- **Time Since a Reference Point**: `df['time_since'] = df['datetime'] - reference_date`
- **Is Weekend Feature**: `df['is_weekend'] = df['weekday'].isin([5,6]).astype(int)`

4. Text Features

- **Tokenization**: `nltk.word_tokenize(df['text'])`
- **TF-IDF Vectorization**: `sklearn.feature_extraction.text.TfidfVectorizer().fit_transform(df['text'])`

By: Waleed Mousa

- **Count Vectorization**: `sklearn.feature_extraction.text.CountVectorizer().fit_transform(df['text'])`
- **N-grams**: `sklearn.feature_extraction.text.CountVectorizer(ngram_range=(1,2)).fit_transform(df['text'])`
- **Sentiment Analysis**: `TextBlob(df['text']).sentiment.polarity`

5. Numerical Features

- **Standardization**: `(df['num_feature'] - df['num_feature'].mean()) / df['num_feature'].std()`
- **Min-Max Normalization**: `(df['num_feature'] - df['num_feature'].min()) / (df['num_feature'].max() - df['num_feature'].min())`
- **Robust Scaling (handling outliers)**: `sklearn.preprocessing.RobustScaler().fit_transform(df[['num_feature']])`
- **Binning**: `pd.cut(df['num_feature'], bins=5, labels=False)`
- **Rank Transformation**: `df['num_feature'].rank()`

6. Feature Interaction

- **Polynomial Features**: `sklearn.preprocessing.PolynomialFeatures().fit_transform(df[['feature1', 'feature2']])`
- **Interaction Between Categorical and Numerical Features**: `df['new_feature'] = df['cat_feature'].astype(str) + '_' + df['num_feature'].astype(str)`
- **Feature Crosses for Categorical Features**: `pd.get_dummies(df['feature1'] + '_' + df['feature2'])`
- **Pairwise Products of Numerical Features**: `df['feature1'] * df['feature2']`
- **Creating Ratios of All Pairwise Features**: `df['feature1'] / df['feature2']`

7. Dimensionality Reduction

- **PCA (Principal Component Analysis)**: `sklearn.decomposition.PCA(n_components=3).fit_transform(df)`
- **t-SNE (t-Distributed Stochastic Neighbor Embedding)**: `sklearn.manifold.TSNE(n_components=2).fit_transform(df)`
- **LDA (Linear Discriminant Analysis)**: `sklearn.discriminant_analysis.LinearDiscriminantAnalysis().fit_transform(X, y)`

- **Feature Agglomeration (Clustering of Features)**: `sklearn.cluster.FeatureAgglomeration().fit_transform(df)`
- **UMAP (Uniform Manifold Approximation and Projection)**: `umap.UMAP().fit_transform(df)`

8. Handling Missing Values

- **Imputation with Mean/Median/Mode**: `sklearn.impute.SimpleImputer(strategy='mean').fit_transform(df[['feature']])`
- **KNN Imputation**: `sklearn.impute.KNNImputer(n_neighbors=5).fit_transform(df)`
- **Iterative Imputer (Multivariate Imputation)**: `sklearn.experimental.enable_iterative_imputer.IterativeImputer().fit_transform(df)`
- **Imputation Using (Group) Median**: `df['feature'] = df.groupby('group')['feature'].transform(lambda x: x.fillna(x.median()))`
- **Creating Missing Value Indicator**: `df['feature_is_missing'] = df['feature'].isnull().astype(int)`

9. Feature Extraction from Unstructured Data

- **Image Feature Extraction (e.g., Color Histograms)**: `cv2.calcHist([image], channels, mask, histSize, ranges)`
- **Text Embeddings (Word2Vec, GloVe)**: `gensim.models.Word2Vec(sentences).wv`
- **Audio Feature Extraction (MFCC)**: `librosa.feature.mfcc(y=audio_waveform, sr=sampling_rate)`
- **Extracting Features from Time Series Data**: `tsfresh.extract_features(time_series, column_id='id')`
- **Image Resizing and Flattening for ML**: `cv2.resize(image, (width, height)).flatten()`
- **Bag of Words for Text Data**: `sklearn.feature_extraction.text.CountVectorizer().fit_transform(corpus)`

10. Geospatial Feature Engineering

- **Latitude and Longitude to Cartesian Coordinates**: `np.radians(df['latitude']), np.radians(df['longitude'])`
- **Haversine Distance Between Two Points**: `haversine(lon1, lat1, lon2, lat2)`
- **Geospatial Clustering (e.g., DBSCAN)**: `sklearn.cluster.DBSCAN(eps=0.01, min_samples=10).fit(geo_coordinates)`

- **Extracting ZIP Codes from Addresses**: `uszipcode.SearchEngine().by_address(address, returns=1)`

11. Time-based Features

- **Elapsed Time Since a Reference Event**: `df['time_since_event'] = pd.to_datetime(df['timestamp']) - reference_timestamp`
- **Cyclical Time Features (e.g., Hour of Day, Day of Week)**: `np.sin(2 * np.pi * df['hour']/23.0), np.cos(2 * np.pi * df['hour']/23.0)`
- **Time to Next/Previous Event**: `df['time_to_next_event'] = df['timestamp'].shift(-1) - df['timestamp']`
- **Window Functions for Time Series (e.g., Rolling Mean)**: `df['rolling_mean'] = df['value'].rolling(window=5).mean()`
- **Exponential Weighted Moving Average**: `df['ewma'] = df['value'].ewm(span=50).mean()`

12. Textual Feature Engineering

- **Named Entity Recognition (NER)**: `spacy.load('en_core_web_sm').ents(text)`
- **Part-of-Speech Tagging**: `nltk.pos_tag(tokens)`
- **Term Frequency-Inverse Document Frequency (TF-IDF)**: `TfidfVectorizer().fit_transform(corpus)`
- **Extracting Readability Features**: `textstat.flesch_reading_ease(text)`
- **Word Embedding (Pre-trained Models like GloVe, BERT)**: `transformers.BertModel.from_pretrained('bert-base-uncased').encode(text)`

13. Feature Selection Techniques

- **SelectKBest with Custom Score Function**: `SelectKBest(score_func=f_regression, k=10).fit_transform(X, y)`
- **Recursive Feature Elimination**: `RFE(estimator, n_features_to_select=10).fit(X, y)`
- **Feature Importance From Tree-based Models**: `model.feature_importances_`
- **L1 Regularization for Feature Selection (Lasso)**: `Lasso(alpha=0.1).fit(X, y)`
- **Variance Thresholding for Feature Selection**: `VarianceThreshold(threshold=(.8 * (1 - .8))).fit_transform(X)`

14. Handling Imbalanced Data

- **Random Over-sampling**: `RandomOverSampler().fit_resample(X, y)`
- **Random Under-sampling**: `RandomUnderSampler().fit_resample(X, y)`

- **SMOTE for Over-sampling**: `SMOTE().fit_resample(X, y)`
- **ADASYN for Adaptive Synthetic Sampling**: `ADASYN().fit_resample(X, y)`

15. Advanced Numerical Techniques

- **Discretization (Binning)**: `KBinsDiscretizer(n_bins=5, encode='ordinal').fit_transform(X)`
- **Log Transform Plus One**: `np.log1p(df['num_feature'])`
- **Square Root Transform**: `np.sqrt(df['num_feature'])`
- **Inverse Transform**: `np.reciprocal(df['num_feature'])`

16. Scaling and Normalization

- **MaxAbsScaler for Scaling**: `MaxAbsScaler().fit_transform(X)`
- **Normalizer for Row-wise Normalization**: `Normalizer().fit_transform(X)`
- **Quantile Transformer for Robust Scaling**: `QuantileTransformer().fit_transform(X)`

17. Encoding High Cardinality Features

- **Target Encoding**: `category_encoders.TargetEncoder().fit_transform(X, y)`
- **Binary Encoding for High Cardinality**: `category_encoders.BinaryEncoder().fit_transform(df['high_card_feature'])`
- **Hashing Trick**: `category_encoders.HashingEncoder().fit_transform(df['high_card_feature'])`

18. Feature Engineering for Time Series

- **Fourier Transform for Periodic Patterns**: `np.fft.fft(time_series)`
- **Lag Features for Autocorrelation**: `df['lag_1'] = df['value'].shift(1)`
- **Rolling Window Features (e.g., Rolling Variance)**: `df['rolling_var'] = df['value'].rolling(window=5).var()`
- **Decomposing Time Series (Seasonal-Trend Decomposition)**: `seasonal_decompose(time_series, model='additive')`

19. Custom Feature Engineering

- **Custom Transformer in Pipeline**: `Pipeline(steps=[('custom_transformer', CustomTransformer()), ...])`
- **Conditional Feature Creation**: `df['new_feature'] = np.where(df['condition'], value_if_true, value_if_false)`

20. Data Reduction for Efficiency

- **Random Projection for Dimensionality Reduction**: `GaussianRandomProjection().fit_transform(X)`
- **Feature Agglomeration for Clustering Features**: `FeatureAgglomeration().fit_transform(X)`

21. Handling Multi-dimensional Data

- **Flattening Image Data for ML Models**: `np.reshape(image_data, (num_samples, -1))`
- **PCA for Image Data Dimensionality Reduction**: `PCA(n_components=0.95).fit_transform(image_data)`

22. Feature Engineering for Specific Models

- **Word Tokenization for NLP Models**: `nltk.word_tokenize(text)`
- **Creating Polynomial Features for Regression Models**: `PolynomialFeatures(degree=2).fit_transform(X)`

23. Integration with External Data

- **Merging External Data Sources**: `pd.merge(df, external_data, on='key')`
- **Feature Engineering from APIs (e.g., Weather Data for Sales Predictions)**: `fetch_weather_data(api_key, location)`

24. Feature Engineering Automation

- **Automated Feature Engineering (Featuretools)**: `featuretools.dfs(entityset=es, target_entity='target')`

25. Features from Aggregations

- **GroupBy Aggregations for Categorical Features**: `df.groupby('category').agg({'num_feature': ['mean', 'sum']})`
- **Window Functions in Pandas**: `df['rolling_mean'] = df.sort_values('time').groupby('group').rolling(window=3)['value'].mean()`

26. Encoding Techniques for Tree-based Models

- **Ordinal Encoding for Tree-Based Models**: `OrdinalEncoder().fit_transform(X)`

- **Frequency Encoding for Tree-Based Models**: `df['freq_encode'] = df.groupby('feature')['feature'].transform('count') / len(df)`

27. Multi-level Feature Engineering

- **Hierarchical Interactions (e.g., City within Country)**: `df['city_country'] = df['city'] + '_' + df['country']`
- **Creating Features from Hierarchical Clustering**: `AgglomerativeClustering(n_clusters=10).fit_predict(X)`

28. Signal Processing Features

- **Fast Fourier Transform for Signal Data**: `np.fft.fft(signal)`
- **Signal Filtering (e.g., Low/High/Band-Pass Filters)**: `scipy.signal.butter(N, Wn, btype, output='sos')`
- **Signal Envelope Extraction**: `scipy.signal.hilbert(signal)`

29. Handling Sparse Data

- **Sparse Matrix Representation**: `scipy.sparse.csr_matrix(data)`
- **Dimensionality Reduction for Sparse Data**: `sklearn.decomposition.TruncatedSVD(n_components).fit_transform(sparse_data)`

30. Feature Hashing

- **Feature Hashing for Vectorizing Text**: `sklearn.feature_extraction.FeatureHasher(n_features).transform(raw_features)`
- **Hashing Trick for Categorical Features**: `sklearn.feature_extraction.text.HashingVectorizer(n_features)`

31. Features from Probabilistic Distributions

- **Extracting Parameters from Distributions**: `scipy.stats.norm.fit(data)`
- **Generating Features from Distribution Samples**: `np.random.normal(loc, scale, size)`

32. Text Specific Features

- **Character Length of Text**: `df['text_length'] = df['text'].apply(len)`

- **Count of Specific Words or Characters**: `df['word_count'] = df['text'].str.count('specific_word')`

33. Image Specific Features

- **Edge Detection (e.g., Sobel, Canny)**: `cv2.Canny(image, threshold1, threshold2)`
- **Feature Descriptors (e.g., SIFT, SURF)**: `cv2.xfeatures2d.SIFT_create().detectAndCompute(image, None)`

34. Multi-Label Feature Engineering

- **Binary Relevance Transformation**: `sklearn.preprocessing.MultiLabelBinarizer().fit_transform(multi_label)`

35. Survival Analysis Features

- **Cox Proportional Hazards Features**: `lifelines.CoxPHFitter().fit(df, duration_col, event_col).predict_partial_hazard(df)`

36. Features from Graphs/Networks

- **Node Centrality Measures (e.g., Degree, Eigenvector)**: `networkx.degree_centrality(G)`
- **Graph Embeddings (e.g., Node2Vec)**: `node2vec = Node2Vec(G); model = node2vec.fit()`

37. Cross-validation in Feature Engineering

- **K-Fold Cross-Validated Features**: `sklearn.model_selection.cross_val_predict(model, X, y, cv=5)`

38. Handling Unstructured Data

- **Converting Unstructured Data to Structured Form**: `extract_features_from_unstructured_data(unstructured_data)`

39. Features from Audio Data

- **Mel-Frequency Cepstral Coefficients (MFCCs)**: `librosa.feature.mfcc(y=audio, sr=sampling_rate)`

- **Spectral Centroid**: `librosa.feature.spectral_centroid(y=audio, sr=sampling_rate)`

40. Custom Feature Engineering Functions

- **User-Defined Transformation Functions**: `df['custom_feature'] = df['feature'].apply(custom_transformation_function)`

41. Feature Engineering in Time Series

- **Lagged Features for Autocorrelation**: `df['lag_feature'] = df['feature'].shift(periods)`
- **Rolling Window Features (e.g., Rolling Std)**: `df['rolling_std'] = df['feature'].rolling(window=5).std()`

42. Natural Language Specific Features

- **Part-of-Speech Tag Counts**: `nltk.pos_tag(text).count('NN') # count of nouns`
- **Text Complexity Features (e.g., Flesch Reading Ease)**: `textstat.flesch_reading_ease(text)`

43. Social Media Specific Features

- **Sentiment Score from Social Media Text**: `TextBlob(text).sentiment.polarity`
- **Social Media Engagement Features (e.g., Likes, Shares)**: `calculate_engagement_metrics(df['likes'], df['shares'])`

44. Geographic Features

- **Distance to Points of Interest**: `calculate_distance_to_poi(lat, long, poi_lat, poi_long)`
- **Geographic Clustering Features**: `DBSCAN(eps=0.01, min_samples=10).fit(geo_data)`

45. Time-based Aggregation Features

- **Cumulative Count or Sum Over Time**: `df.groupby('time').cumcount()`
- **Time-based Aggregation (e.g., Sum of Sales per Month)**: `df.groupby(pd.Grouper(key='date', freq='M'))['sales'].sum()`

46. Online Behavior Features

- **Session Duration in Web Analytics**: `calculate_session_duration(df['session_start'], df['session_end'])`
- **Click Through Rate (CTR) for Online Ads**: `df['CTR'] = df['clicks'] / df['impressions']`

47. Biomedical Signal Features

- **Heart Rate Variability Metrics**: `calculate_hrv(ecg_signal)`
- **Biomedical Image Texture Features**: `skimage.feature.greycomatrix(image, [distance], [angle])`

48. Feature Pipelines

- **Creating a Feature Pipeline**: `Pipeline(steps=[('feature_creation', CustomFeatureTransformer()), ...])`