[Data Modeling] {CheatSheet}

1. Feature Selection and Engineering

- **Removing Irrelevant Features**: `df.drop(['irrelevant_column'], axis=1, inplace=True)`
- **Correlation Matrix for Feature Selection**: `corr_matrix = df.corr()`
- **Univariate Selection with SelectKBest**: `from sklearn.feature_selection import SelectKBest; X_new = SelectKBest(k=2).fit_transform(X, y)`
- **Recursive Feature Elimination**: `from sklearn.feature_selection import RFE; rfe = RFE(estimator, n_features_to_select=5); X_rfe = rfe.fit_transform(X, y)`
- **Principal Component Analysis (PCA)**: `from sklearn.decomposition import PCA; pca = PCA(n_components=2); X_pca = pca.fit_transform(X)`

2. Data Preprocessing

- **Standard Scaling**: `from sklearn.preprocessing import StandardScaler; scaler = StandardScaler(); X_scaled = scaler.fit_transform(X)`
- **Normalizing Data**: `from sklearn.preprocessing import Normalizer; normalizer = Normalizer(); X_normalized = normalizer.transform(X)`
- **Encoding Categorical Variables**: `pd.get_dummies(df)`
- **Handling Missing Values**: `from sklearn.impute import SimpleImputer; imputer = SimpleImputer(strategy='mean'); X_imputed = imputer.fit_transform(X)`
- **Polynomial Feature Generation**: `from sklearn.preprocessing import PolynomialFeatures; poly = PolynomialFeatures(degree=2); X_poly = poly.fit_transform(X)`

3. Model Selection

- **Linear Regression**: `from sklearn.linear_model import LinearRegression; model = LinearRegression()`
- **Logistic Regression**: `from sklearn.linear_model import LogisticRegression; model = LogisticRegression()`
- **Decision Trees**: `from sklearn.tree import DecisionTreeClassifier; model = DecisionTreeClassifier()`

By: Waleed Mousa

- **Random Forest**: `from sklearn.ensemble import RandomForestClassifier; model = RandomForestClassifier()`
- **Support Vector Machines**: `from sklearn.svm import SVC; model = SVC()`

4. Model Training

- **Training a Model**: `model.fit(X_train, y_train)`
- **Cross-Validation**: `from sklearn.model_selection import cross_val_score; scores = cross_val_score(model, X, y, cv=5)`
- **Grid Search for Hyperparameter Tuning**: `from sklearn.model_selection import GridSearchCV; param_grid = {'param': [values]}; grid_search = GridSearchCV(model, param_grid, cv=5); grid_search.fit(X, y)`
- **Random Search for Hyperparameter Tuning**: `from sklearn.model_selection import RandomizedSearchCV; param_distributions = {'param': [values]}; random_search = RandomizedSearchCV(model, param_distributions, n_iter=50, cv=5); random_search.fit(X, y)`

5. Model Evaluation

- **Accuracy Score**: `from sklearn.metrics import accuracy_score; accuracy = accuracy_score(y_true, y_pred)`
- **Confusion Matrix**: `from sklearn.metrics import confusion_matrix; cm = confusion_matrix(y_true, y_pred)`
- **ROC-AUC Score**: `from sklearn.metrics import roc_auc_score; roc_auc = roc_auc_score(y_true, y_scores)`
- **Mean Squared Error**: `from sklearn.metrics import mean_squared_error; mse = mean_squared_error(y_true, y_pred)`
- **Cross-Validation Score**: `from sklearn.model_selection import cross_val_score; cv_scores = cross_val_score(model, X, y, cv=5)`

6. Model Validation and Selection

- **Train-Test Split**: `from sklearn.model_selection import train_test_split; X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)`

- **K-Fold Cross-Validation**: `from sklearn.model_selection import KFold; kf = KFold(n_splits=5); for train_index, test_index in kf.split(X): ...`
- **Leave-One-Out Cross-Validation**: `from sklearn.model_selection import LeaveOneOut; loo = LeaveOneOut(); for train_index, test_index in loo.split(X): ...`
- **Stratified Sampling**: `from sklearn.model_selection import StratifiedShuffleSplit; sss = StratifiedShuffleSplit(n_splits=5, test_size=0.5); for train_index, test_index in sss.split(X, y): ...`

7. Ensemble Methods

- **Bagging with Random Forest**: `from sklearn.ensemble import RandomForestClassifier; model = RandomForestClassifier(n_estimators=100)`
- **Boosting with XGBoost**: `from xgboost import XGBClassifier; model = XGBClassifier(n_estimators=100, learning_rate=0.1)`
- **AdaBoost**: `from sklearn.ensemble import AdaBoostClassifier; model = AdaBoostClassifier(n_estimators=100)`
- **Gradient Boosting**: `from sklearn.ensemble import GradientBoostingClassifier; model = GradientBoostingClassifier(n_estimators=100)`

8. Neural Networks and Deep Learning

- **Basic Neural Network with Keras**: `from keras.models import Sequential; from keras.layers import Dense; model = Sequential(); model.add(Dense(units=64, activation='relu', input_dim=100)); model.add(Dense(units=10, activation='softmax'))`
- **Compiling a Keras Model**: `model.compile(loss='categorical_crossentropy', optimizer='sgd', metrics=['accuracy'])`
- **Training a Keras Model**: `model.fit(X_train, y_train, epochs=5, batch_size=32)`
- **Using Pretrained Models in Keras**: `from keras.applications import VGG16; model = VGG16(weights='imagenet')`

9. Clustering and Unsupervised Learning

- **K-Means Clustering**: `from sklearn.cluster import KMeans; kmeans = KMeans(n_clusters=3); kmeans.fit(X)`
- **Hierarchical Clustering**: `from sklearn.cluster import AgglomerativeClustering; clustering = AgglomerativeClustering().fit(X)`
- **DBSCAN for Density-Based Clustering**: `from sklearn.cluster import DBSCAN; clustering = DBSCAN(eps=3, min_samples=2).fit(X)`
- **PCA for Dimensionality Reduction in Clustering**: `from sklearn.decomposition import PCA; pca = PCA(n_components=2); X_pca = pca.fit_transform(X)`

10. Time Series Analysis

- **ARIMA Model for Time Series**: `from statsmodels.tsa.arima_model import ARIMA; model = ARIMA(time_series, order=(5,1,0)); model_fit = model.fit(disp=0)`
- **Seasonal Decomposition**: `from statsmodels.tsa.seasonal import seasonal_decompose; decomposition = seasonal_decompose(time_series)`
- **Using Prophet for Forecasting**: `from fbprophet import Prophet; m = Prophet(); m.fit(df); future = m.make_future_dataframe(periods=365); forecast = m.predict(future)`

11. Natural Language Processing (NLP)

- **Bag of Words Model**: `from sklearn.feature_extraction.text import CountVectorizer; vectorizer = CountVectorizer(); X = vectorizer.fit_transform(corpus)`
- **TF-IDF Vectorization**: `from sklearn.feature_extraction.text import TfidfVectorizer; vectorizer = TfidfVectorizer(); X = vectorizer.fit_transform(corpus)`
- **Word Embeddings with Word2Vec**: `from gensim.models import Word2Vec; model = Word2Vec(sentences, size=100, window=5, min_count=1, workers=4)`
- **Sentiment Analysis**: `from textblob import TextBlob; polarity, subjectivity = TextBlob(text).sentiment`

12. Image and Video Analysis

- **Image Classification with CNNs**: `from keras.layers import Conv2D, MaxPooling2D; model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(64, 64, 3))); model.add(MaxPooling2D(pool_size=(2, 2)))`
- **Object Detection with Pretrained Models**: `from imageai.Detection import ObjectDetection; detector = ObjectDetection(); detections = detector.detectObjectsFromImage(input_image="image.jpg", output_image_path="image_new.jpg")`
- **Video Processing with OpenCV**: `import cv2; cap = cv2.VideoCapture('video.mp4'); while cap.isOpened(): ret, frame = cap.read(); if not ret: break; processed_frame = process(frame); cv2.imshow('Frame', processed_frame)`

13. Reinforcement Learning

- **Q-Learning**: `import numpy as np; Q = np.zeros([env.observation_space.n, env.action_space.n]); for episode in range(1, num_episodes): state = env.reset(); done = False; while not done: action = np.argmax(Q[state] + np.random.randn(1, env.action_space.n) * (1./(episode+1)))`
- **Deep Q-Network (DQN)**: `from rl.agents.dqn import DQNAgent; dqn = DQNAgent(model=model, policy=policy, memory=memory, nb_actions=nb_actions)`

14. Hyperparameter Tuning

- **Grid Search in scikit-learn**: `from sklearn.model_selection import GridSearchCV; grid_search = GridSearchCV(estimator, param_grid, cv=3)`
- **Random Search in scikit-learn**: `from sklearn.model_selection import RandomizedSearchCV; random_search = RandomizedSearchCV(estimator, param_distributions, n_iter=100, cv=3)`

15. Model Evaluation Metrics

- **Classification Report**: `from sklearn.metrics import classification_report; print(classification_report(y_true, y_pred))`

- **Mean Absolute Error (MAE)**: `from sklearn.metrics import mean_absolute_error; mae = mean_absolute_error(y_true, y_pred)`
- **F1 Score**: `from sklearn.metrics import f1_score; f1 = f1_score(y_true, y_pred)`
- **Silhouette Score (Clustering)**: `from sklearn.metrics import silhouette_score; silhouette = silhouette_score(X, labels)`

16. Model Persistence and Serialization

- **Save Model with joblib**: `from joblib import dump; dump(model, 'model.joblib')`
- **Load Model with joblib**: `from joblib import load; model = load('model.joblib')`
- **Save Keras Model**: `model.save('model.h5')`
- **Load Keras Model**: `from keras.models import load_model; model = load_model('model.h5')`

17. Advanced Model Types

- **Multiclass Classification**: `from sklearn.multiclass import OneVsRestClassifier; model = OneVsRestClassifier(estimator)`
- **Multioutput Regression**: `from sklearn.multioutput import MultiOutputRegressor; model = MultiOutputRegressor(estimator)`
- **Stacking Models**: `from sklearn.ensemble import StackingClassifier; stack_model = StackingClassifier(estimators=base_learners, final_estimator=final_estimator)`

18. Dealing with Imbalanced Data

- **Random Over-Sampling**: `from imblearn.over_sampling import RandomOverSampler; ros = RandomOverSampler(); X_res, y_res = ros.fit_resample(X, y)`
- **Random Under-Sampling**: `from imblearn.under_sampling import RandomUnderSampler; rus = RandomUnderSampler(); X_res, y_res = rus.fit_resample(X, y)`
- **SMOTE for Over-Sampling**: `from imblearn.over_sampling import SMOTE; smote = SMOTE(); X_res, y_res = smote.fit_resample(X, y)`

19. Model Explainability and Interpretation

- **Feature Importance with Random Forest**: `importances = model.feature_importances_`
- **SHAP Values**: `import shap; explainer = shap.TreeExplainer(model); shap_values = explainer.shap_values(X)`
- **Permutation Importance**: `from sklearn.inspection import permutation_importance; perm_importance = permutation_importance(model, X_val, y_val)`
- **Partial Dependence Plots**: `from sklearn.inspection import plot_partial_dependence; plot_partial_dependence(model, X, [features_indices])`

20. Time Series Forecasting

- **ARIMA Model**: `from statsmodels.tsa.arima.model import ARIMA; model = ARIMA(ts, order=(p, d, q)); model_fit = model.fit()`
- **Seasonal Decomposition**: `from statsmodels.tsa.seasonal import seasonal_decompose; result = seasonal_decompose(ts, model='additive')`
- **Exponential Smoothing**: `from statsmodels.tsa.holtwinters import ExponentialSmoothing; model = ExponentialSmoothing(ts, trend='add', seasonal='add', seasonal_periods=12)`

21. Text Analysis and NLP Models

- **Count Vectorizer**: `from sklearn.feature_extraction.text import CountVectorizer; vect = CountVectorizer(); X_vect = vect.fit_transform(corpus)`
- **TF-IDF Transformer**: `from sklearn.feature_extraction.text import TfidfTransformer; tfidf = TfidfTransformer(); X_tfidf = tfidf.fit_transform(X_vect)`
- **Word Embeddings with Gensim**: `from gensim.models import Word2Vec; model = Word2Vec(sentences, vector_size=100, window=5, min_count=1, workers=4)`
- **BERT Embeddings**: `from sentence_transformers import SentenceTransformer; model = SentenceTransformer('bert-base-nli-mean-tokens'); embeddings = model.encode(sentences)`

22. Image Processing and Computer Vision

- **Image Augmentation with Keras**: `from keras.preprocessing.image import ImageDataGenerator; datagen = ImageDataGenerator(rotation_range=40, width_shift_range=0.2)`
- **Pre-trained Models (e.g., VGG, ResNet)**: `from keras.applications.vgg16 import VGG16; model = VGG16(weights='imagenet')`
- **OpenCV for Image Preprocessing**: `import cv2; image = cv2.imread('path/to/image.jpg'); gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)`

23. Collaborative Filtering and Recommendation Systems

- **Matrix Factorization with SVD for Recommendations**: `from surprise import SVD; algo = SVD(); algo.fit(data)`
- **K-Nearest Neighbors for Collaborative Filtering**: `from surprise import KNNBasic; algo = KNNBasic(); algo.fit(data)`
- **Creating a User-Item Interaction Matrix**: `interaction_matrix = df.pivot(index='user_id', columns='item_id', values='rating')`

24. Anomaly Detection

- **Isolation Forest for Anomaly Detection**: `from sklearn.ensemble import IsolationForest; iso_forest = IsolationForest(); anomalies = iso_forest.fit_predict(X)`
- **DBSCAN for Density-Based Anomaly Detection**: `from sklearn.cluster import DBSCAN; dbscan = DBSCAN(); cluster_labels = dbscan.fit_predict(X)`
- **Local Outlier Factor**: `from sklearn.neighbors import LocalOutlierFactor; lof = LocalOutlierFactor(); outliers = lof.fit_predict(X)`

25. Dimensionality Reduction

- **t-SNE for Dimensionality Reduction**: `from sklearn.manifold import TSNE; X_reduced = TSNE(n_components=2).fit_transform(X)`
- **PCA (Principal Component Analysis)**: `from sklearn.decomposition import PCA; pca = PCA(n_components=3); X_pca = pca.fit_transform(X)`

- **LDA (Linear Discriminant Analysis)**: `from sklearn.discriminant_analysis import LinearDiscriminantAnalysis; lda = LinearDiscriminantAnalysis(); X_lda = lda.fit_transform(X, y)`

26. Sequential Data and Time Series

- **LSTM for Sequence Data**: `from keras.models import Sequential; from keras.layers import LSTM; model = Sequential(); model.add(LSTM(50, return_sequences=True, input_shape=(time_steps, n_features)))`
- **GRU for Time Series**: `from keras.layers import GRU; model.add(GRU(units=50, return_sequences=True))`
- **Time Series Split in Cross-Validation**: `from sklearn.model_selection import TimeSeriesSplit; tscv = TimeSeriesSplit(n_splits=5)`

27. Graph Models

- **NetworkX for Graph Analysis**: `import networkx as nx; G = nx.from_pandas_edgelist(df, 'source', 'target'); nx.draw(G)`
- **Graph Convolutional Networks**: `import torch_geometric.nn as geom_nn; conv = geom_nn.GCNConv(in_channels, out_channels)`

28. Survival Analysis

- **Cox Proportional Hazards Model**: `from lifelines import CoxPHFitter; cph = CoxPHFitter(); cph.fit(df, duration_col='T', event_col='E')`
- **Kaplan-Meier Estimator**: `from lifelines import KaplanMeierFitter; kmf = KaplanMeierFitter(); kmf.fit(durations, event_observed)`

29. Advanced Feature Engineering

- **Feature Interaction**: `df['interaction'] = df['feature1'] * df['feature2']`
- **Lag Features for Time Series**: `df['lag_feature'] = df['feature'].shift(periods=1)`