Probability 2

Certainly! Let's go through examples of Binomial Distribution, Poisson Distribution, building a Normal Q-Q Plot, interpreting the Q-Q Plot, understanding the Central Limit Theorem (CLT) for sampling variations, computing and analyzing Confidence Intervals, and basic techniques for Data Cleansing (Dealing with Missing Data, Outlier Detection).

1. Binomial Distribution with NumPy and Matplotlib:

pythonCopy code

```
from scipy.stats import binom # Parameters for the binomial distribution n = 10 # Number of trials p = 0.3 # Probability of success # Generate data for a binomial distribution data_binomial = binom.rvs(n, p, size=1000) # Create a histogram plt.hist(data_binomial, bins=np.arange(0, n+2)-0.5, color='purple', alpha=0.7) plt.title('Binomial Distribution') plt.xlabel('Number of Successes') plt.ylabel('Frequency') plt.show()
```

2. Poisson Distribution with NumPy and Matplotlib:

pythonCopy code

```
from scipy.stats import poisson # Parameter for the Poisson distribution lambda_param = 3.5 # Generate data for a Poisson distribution data_poisson = poisson.rvs(lambda_param, size=1000) # Create a histogram plt.hist(data_poisson, bins=30, color='brown', alpha=0.7) plt.title('Poisson Distribution') plt.xlabel('Number of Events') plt.ylabel('Frequency') plt.show()
```

3. Building Normal Q-Q Plot and Interpretation:

pythonCopy code

```
from scipy.stats import probplot # Generate data for a normal Q-Q plot data_normal_qq = np.random.normal(size=100) # Create a normal Q-Q plot probplot(data_normal_qq, plot=plt) plt.title('Normal Q-Q Plot') plt.show()
```

4. Central Limit Theorem (CLT) for Sampling Variations:

The Central Limit Theorem states that, for a sufficiently large sample size, the distribution of the sample mean will be approximately normally distributed, regardless of the shape of the population distribution.

pythonCopy code

```
# Generate data from a non-normal distribution data_non_normal = np.random.exponential(size=1000) # Calculate means of random samples sample_means = [np.mean(np.random.choice(data_non_normal, size=30)) for _ in range(1000)] # Create a histogram of sample means plt.hist(sample_means, bins=30, color='green', alpha=0.7) plt.title('Central Limit Theorem') plt.xlabel('Sample Means') plt.ylabel('Frequency') plt.show()
```

5. Confidence Interval Computation and Analysis:

pythonCopy code

```
# Generate data for a sample data_sample = np.random.normal(loc=50, scale=10, size=100) # Calculate the confidence interval for the mean confidence_interval = np.percentile(data_sample, [2.5, 97.5]) print(f"Confidence Interval: {confidence_interval}")
```

6. Data Cleansing (Dealing with Missing Data, Outlier Detection):

pythonCopy code

```
# Generate a DataFrame with missing values df_missing = pd.DataFrame({'A': [1, 2, np.nan, 4, 5], 'B': [np.nan, 2, 3, 4, 5]}) # Drop rows with missing values df_missing_cleaned = df_missing.dropna() # Detect outliers using z-score z_scores = np.abs((df_missing_cleaned - df_missing_cleaned.mean()) / df_missing_cleaned.std()) outliers = (z_scores > 3).any(axis=1) # Remove outliers df_cleaned_outliers = df_missing_cleaned[~outliers] print("Original DataFrame:") print(df_missing) print("\nCleaned DataFrame (missing values removed):") print(df_missing_cleaned) print("\nCleaned DataFrame (outliers removed):") print(df_cleaned_outliers)
```

These examples cover various statistical concepts and techniques using NumPy, SciPy, Matplotlib, and Pandas. Customize the code as needed for your specific use case and data.

2. 3rd part

Certainly! Let's go through examples of various feature engineering and data transformation techniques using Python libraries such as Pandas and Scikit-Learn.

1. Label Encoding and One-Hot Encoding with Pandas:

pythonCopy code

```
import pandas as pd from sklearn.preprocessing import LabelEncoder, OneHotEncoder # Create a sample DataFrame data = {'Category': ['A', 'B', 'A', 'C', 'B']} df = pd.DataFrame(data) # Label Encoding label_encoder = LabelEncoder() df['Category_LabelEncoded'] = label_encoder.fit_transform(df['Category']) # One-Hot Encoding onehot_encoder = OneHotEncoder(sparse=False, drop='first') onehot_encoded = onehot_encoder.fit_transform(df[['Category']]) df_onehot = pd.DataFrame(onehot_encoded, columns=['Category_B', 'Category_C']) # Concatenate the one-hot encoded DataFrame to the original DataFrame df = pd.concat([df, df_onehot], axis=1) print(df)
```

2. Data Transformation (Merging, Ordering, Aggregation):

pythonCopy code

```
# Merge DataFrames df1 = pd.DataFrame({'ID': [1, 2, 3], 'Value1': [10, 20, 30]}) df2 = pd.DataFrame({'ID': [2, 3, 4], 'Value2': [40, 50, 60]}) df_merged = pd.merge(df1, df2, on='ID', how='outer') # Order DataFrame by a column df_ordered = df_merged.sort_values(by='ID') # Aggregation df_aggregated = df_merged.groupby('ID').agg({'Value1': 'sum', 'Value2': 'mean'}).reset_index() print(df_merged) print(df_ordered) print(df_aggregated)
```

3. Data Sampling (Balanced, Stratified):

pythonCopy code

```python
from sklearn.model_selection import train_test_split # Balanced Sampling df_balanced = df.groupby('Category_LabelEncoded', group_keys=False).apply(lambda x: x.sample(min(len(x), 2))) # Stratified Sampling X_train, X_test, y_train, y_test = train_test_split(df[['Category_B', 'Category_C']], df['Category_LabelEncoded'], test_size=0.2, stratify=df['Category_LabelEncoded']) print(df_balanced) print(X_train, X_test, y_train, y_test)
```

4. Data Partitioning (Create Training + Validation + Test Data Set):

pythonCopy code

```python
# Split data into training, validation, and test sets X_train, X_temp, y_train, y_temp = train_test_split(df[['Category_B', 'Category_C']], df['Category_LabelEncoded'], test_size=0.4, random_state=42) X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42) print(X_train.shape, X_val.shape, X_test.shape)
```

5. Data Transformations (Normalization, Standardization, Scaling):

pythonCopy code

```python
from sklearn.preprocessing import MinMaxScaler, StandardScaler # Normalization minmax_scaler = MinMaxScaler() X_normalized = minmax_scaler.fit_transform(X_train) # Standardization standard_scaler = StandardScaler() X_standardized = standard_scaler.fit_transform(X_train) print(X_normalized) print(X_standardized)
```

6. Binning (Count-Based, Handling Of Missing Values as its own Group):

pythonCopy code

```python
# Count-Based Binning df['Value_Binned'] = pd.cut(df['Value1'], bins=[0, 10, 20, 30], labels=['Low', 'Medium', 'High']) # Handling Missing Values as its own Group df['Category'] = df['Category'].fillna('Unknown') print(df)
```

7. Data Replacement (Cutting, Splitting, Merging):

pythonCopy code

```
# Cutting values into discrete intervals df['Value_Cut'] = pd.cut(df['Value1'], bins=[0, 10, 20, 30], labels=['Interval1', 'Interval2', 'Interval3']) # Splitting and Merging df_split = df['Category'].str.split(',', expand=True) df_merged = pd.concat([df, df_split], axis=1) print(df) print(df_split) print(df_merged)
```

8. Weighting And Selection (Attribute Weighting, Automatic Optimization):

pythonCopy code

```
# Attribute Weighting (e.g., using feature importance from a model) from sklearn.ensemble import RandomForestClassifier model = RandomForestClassifier() model.fit(X_train, y_train) feature_importances = model.feature_importances_ # Automatic Optimization (e.g., hyperparameter tuning) from sklearn.model_selection import GridSearchCV from sklearn.svm import SVC param_grid = {'C': [0.1, 1, 10], 'gamma': [0.01, 0.1, 1]} grid_search = GridSearchCV(SVC(), param_grid, cv=5) grid_search.fit(X_train, y_train) best_params = grid_search.best_params_ print(feature_importances) print(best_params)
```

9. Imputation (Replacement of Missing Observations with Statistical Algorithms):

pythonCopy code

```
from sklearn.impute import SimpleImputer # Replace missing values with mean imputer = SimpleImputer(strategy='mean') X_imputed = imputer.fit_transform(X_train) print(X_imputed)
```

These examples cover a variety of feature engineering and data transformation techniques using Pandas and Scikit-Learn. Customize the code as needed for your specific use case and data.