Below is the implementation of tasks for **Personalized Cancer Diagnosis** (Weeks 4-8). This includes structured code, explanations, and solutions for each task.

---

1. Business Problem Overview

- **Objective:** Predict the class of genetic variants based on text data for cancer diagnosis.
- **Constraints:**
  - Interpretability is important for clinical acceptance.
  - Model performance must prioritize precision for certain classes.
  - No real-time latency requirements.

---

2. DS Problem Formulation

Mapping Real World to ML Problem

- **Data:** Genetic data including gene, variation, and related clinical text.
- **ML Problem:** Multiclass classification with 9 classes.
- **Evaluation Metric:** Log Loss (due to probabilistic outputs).

Data Preparation and Splitting

```python
# Import libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

# Load dataset (replace with actual path)
data = pd.read_csv("cancer_variants.csv")

# Combine text features for input
data['combined_text'] = data['Gene'] + " " + data['Variation'] + " " +
data['Text']

# Encode target classes
le = LabelEncoder()
data['target'] = le.fit_transform(data['Class'])
```

```
# Split data into train, CV (validation), and test sets
X = data['combined_text']
y = data['target']
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.4,
random_state=42)
X_cv, X_test, y_cv, y_test = train_test_split(X_temp, y_temp, test_size=0.5,
random_state=42)
```

---

3. Exploratory Data Analysis

Reading and Preprocessing

```
# Display basic dataset information
print(data.head())
print(data.info())

# Check class distribution
class_distribution = data['Class'].value_counts()
print(class_distribution)
```

Random Model for Baseline

```
import numpy as np
from sklearn.metrics import log_loss

# Random predictions for baseline
random_probs = np.random.rand(len(y_cv), len(le.classes_))
logloss_random = log_loss(y_cv, random_probs)
print(f"Baseline Log Loss (Random Model): {logloss_random}")
```

---

4. Univariate Analysis

Gene Feature

```
# Gene distribution
gene_counts = data['Gene'].value_counts()
print("Top Genes:", gene_counts.head())

# Encode gene as a feature using one-hot encoding
gene_onehot = pd.get_dummies(data['Gene'], prefix='Gene')
```

Variation Feature

```python
# Variation distribution
variation_counts = data['Variation'].value_counts()
print("Top Variations:", variation_counts.head())

# Encode variation as a feature using response coding (frequency encoding)
variation_response =
data['Variation'].map(data['Variation'].value_counts(normalize=True))
```

Text Feature

```python
from sklearn.feature_extraction.text import TfidfVectorizer

# Use TF-IDF to vectorize text data
tfidf = TfidfVectorizer(max_features=5000)
X_text = tfidf.fit_transform(data['combined_text'])
```

---

5. Machine Learning Models

Data Preparation

```python
from scipy.sparse import hstack

# Combine features (gene, variation, text)
X_combined = hstack([gene_onehot, variation_response.values[:, None], X_text])
```

Baseline Model: Naive Bayes

```python
from sklearn.naive_bayes import MultinomialNB

# Train Naive Bayes Model
nb = MultinomialNB()
nb.fit(X_combined[:len(X_train)], y_train)

# Evaluate on CV set
y_cv_pred_nb = nb.predict_proba(X_combined[len(X_train):len(X_train) +
len(X_cv)])
logloss_nb = log_loss(y_cv, y_cv_pred_nb)
print(f"Naive Bayes Log Loss: {logloss_nb}")
```

K-Nearest Neighbors (KNN)

```
from sklearn.neighbors import KNeighborsClassifier

# Train KNN model
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_combined[:len(X_train)], y_train)

# Evaluate on CV set
y_cv_pred_knn = knn.predict_proba(X_combined[len(X_train):len(X_train) +
len(X_cv)])
logloss_knn = log_loss(y_cv, y_cv_pred_knn)
print(f"KNN Log Loss: {logloss_knn}")
```

Logistic Regression

```
from sklearn.linear_model import LogisticRegression

# Logistic Regression with class balancing
lr_bal = LogisticRegression(max_iter=1000, class_weight="balanced")
lr_bal.fit(X_combined[:len(X_train)], y_train)

y_cv_pred_lr_bal = lr_bal.predict_proba(X_combined[len(X_train):len(X_train) +
len(X_cv)])
logloss_lr_bal = log_loss(y_cv, y_cv_pred_lr_bal)
print(f"Logistic Regression (Balanced) Log Loss: {logloss_lr_bal}")

# Logistic Regression without class balancing
lr_unbal = LogisticRegression(max_iter=1000)
lr_unbal.fit(X_combined[:len(X_train)], y_train)

y_cv_pred_lr_unbal =
lr_unbal.predict_proba(X_combined[len(X_train):len(X_train) + len(X_cv)])
logloss_lr_unbal = log_loss(y_cv, y_cv_pred_lr_unbal)
print(f"Logistic Regression (Unbalanced) Log Loss: {logloss_lr_unbal}")
```

Linear SVM

```
from sklearn.svm import LinearSVC
from sklearn.calibration import CalibratedClassifierCV

# SVM with calibration for probability outputs
svm = LinearSVC(class_weight="balanced", max_iter=1000)
calibrated_svm = CalibratedClassifierCV(svm)
calibrated_svm.fit(X_combined[:len(X_train)], y_train)

y_cv_pred_svm =
calibrated_svm.predict_proba(X_combined[len(X_train):len(X_train) + len(X_cv)])
logloss_svm = log_loss(y_cv, y_cv_pred_svm)
print(f"Linear SVM Log Loss: {logloss_svm}")
```

Random Forest

```
from sklearn.ensemble import RandomForestClassifier

# Random Forest with one-hot encoded features
rf = RandomForestClassifier(n_estimators=100, class_weight="balanced",
random_state=42)
rf.fit(X_combined[:len(X_train)], y_train)

y_cv_pred_rf = rf.predict_proba(X_combined[len(X_train):len(X_train) +
len(X_cv)])
logloss_rf = log_loss(y_cv, y_cv_pred_rf)
print(f"Random Forest Log Loss: {logloss_rf}")
```

Stacking Classifier

```
from sklearn.ensemble import StackingClassifier

# Stacking with Logistic Regression as meta-classifier
estimators = [('nb', nb), ('knn', knn), ('rf', rf)]
stack = StackingClassifier(estimators=estimators,
final_estimator=LogisticRegression())
stack.fit(X_combined[:len(X_train)], y_train)

y_cv_pred_stack = stack.predict_proba(X_combined[len(X_train):len(X_train) +
len(X_cv)])
logloss_stack = log_loss(y_cv, y_cv_pred_stack)
print(f"Stacking Classifier Log Loss: {logloss_stack}")
```

Majority Voting Classifier

```
from sklearn.ensemble import VotingClassifier

# Voting Classifier (hard voting for simplicity)
voting = VotingClassifier(estimators=estimators, voting='soft')
voting.fit(X_combined[:len(X_train)], y_train)

y_cv_pred_voting = voting.predict_proba(X_combined[len(X_train):len(X_train) +
len(X_cv)])
logloss_voting = log_loss(y_cv, y_cv_pred_voting)
print(f"Voting Classifier Log Loss: {logloss_voting}")
```

---

6. Results Comparison

```
results = pd.DataFrame({
    "Model": ["Naive Bayes", "KNN", "Logistic Regression (Balanced)",
              "Logistic Regression (Unbalanced)", "Linear SVM",
              "Random Forest", "Stacking", "Voting"],
    "Log Loss": [logloss_nb, logloss_knn, logloss_lr_bal, logloss_lr_unbal,
                 logloss_svm, logloss_rf, logloss_stack, logloss_voting]
})
print(results)
```

This comprehensive code implements all tasks listed, from data preparation to advanced machine learning models for cancer diagnosis. Let me know if you need further clarifications or enhancements!