Q5 readme

a. Data Preparation

- **Data Exploration:** Perform missing value statistics, duplicate value checks and unique value counts.
- **Data Preprocessing:** Missing values, outliers and creating new features
  - Fill missing values using the mean
  - Remove obvious outliers in eyesight(left) and eyesight(right) (greater than 2.0)
  - Calculate BMI and create new features based on specific binning rules (e.g., Triglyceride_Rank and Cho_Rank)
  - Create the Cholesterol_Lipit ratio feature
  - Create binary risk features (highrisk and lowhdl) on age and HDL levels.
  - Numeric features are identified and processed for outliers by setting values at the 1% and 95% quantiles to limit extremes.

b. Model Building

Class_counts is calculated by using y_train.value_counts() and due to the class imbalance, later we will apply scale_pos_weight parameter(class weighting) in the models.

For each model, the ROC AUC score is calculated to evaluate performance. And the output is in probabilities.

- **XGBoost and LightGBM**
  - **XGBoost (Extreme Gradient Boosting)**

XGBoost is an efficient implementation of the gradient boosting framework. It has fast learning capabilities and can automatically handle missing values.

```
import xgboost as xgb
from sklearn.metrics import roc_auc_score
xgb_model = xgb.XGBClassifier(n_estimators=600, colsample_bytree=0.7, max_depth=5, learning_rate=0.
    1, random_state=42, scale_pos_weight=1.4)
xgb_model.fit(X_train, y_train)
y_pred_xgb = xgb_model.predict_proba(X_test)[:, 1]
roc_score_xgb = roc_auc_score(y_test, y_pred_xgb)
print(f"ROC AUC Score with XGBoost: {roc_score_xgb}")
#output: ROC AUC Score with XGBoost: 0.87070475126009
```

  - **LightGBM (Light Gradient Boosting Machine)**

LightGBM is an efficient tree-based learning algorithm. It uses a histogram-based algorithm that can reduce memory usage and increase computational speed.

```
import lightgbm as lgb
lgbm_model = lgb.LGBMClassifier(n_estimators=600, max_depth=5, learning_rate=0.1, random_state=42,
    scale_pos_weight=1.4)
lgbm_model.fit(X_train, y_train)
y_pred_proba_lgbm = lgbm_model.predict_proba(X_test)[:, 1]
roc_score_lgbm = roc_auc_score(y_test, y_pred_proba_lgbm)
print(f"ROC AUC Score with LightGBM: {roc_score_lgbm}")
#output: ROC AUC Score with LightGBM: 0.8696467604174507
```

- **VotingClassifier**

  The VotingClassifier is an ensemble method that combines the predictions of multiple different models. It makes the final decision by "voting" on the predictions of all base models, with the option of "hard" or "soft" voting. And it can leverage the strengths of different models, reduce the risk of overfitting and improve the generalizability of the model. In soft voting, the final prediction is a weighted average based on probabilities, whereas in hard voting, it is based on the majority vote.

```
from sklearn.ensemble import VotingClassifier
voting = VotingClassifier(estimators=[('lgbm', lgbm_model),  ('xgb', xgb_model)], voting='soft')
voting.fit(X_train,y_train)
voting_pred = voting.predict_proba(X_test)[:,1]
print('Accuracy: ', roc_auc_score(y_test, voting_pred))
#output:Accuracy:  0.8710017208676449
```

- **Output**

  The VotingClassifier is used to predict the results.

	id	smoking
0	159256	0.699161
1	159257	0.339876
2	159258	0.472245
3	159259	0.027471
4	159260	0.609619
...	...	...
106166	265422	0.676401
106167	265423	0.725054
106168	265424	0.518584
106169	265425	0.141441
106170	265426	0.041142