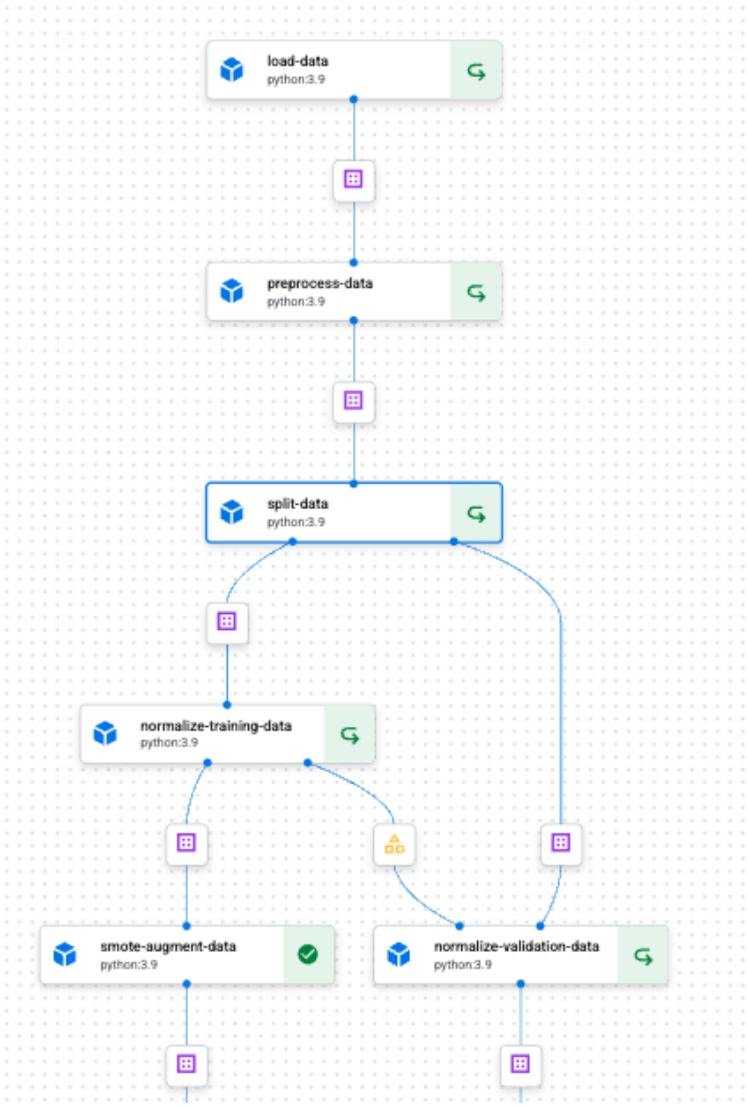
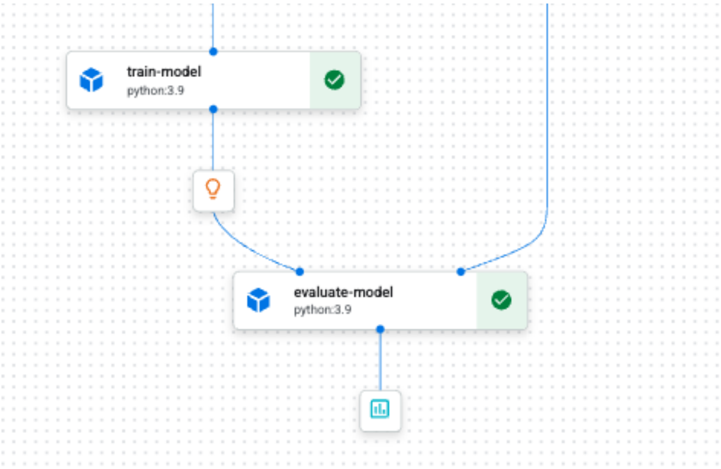
PIPELINE DESIGN

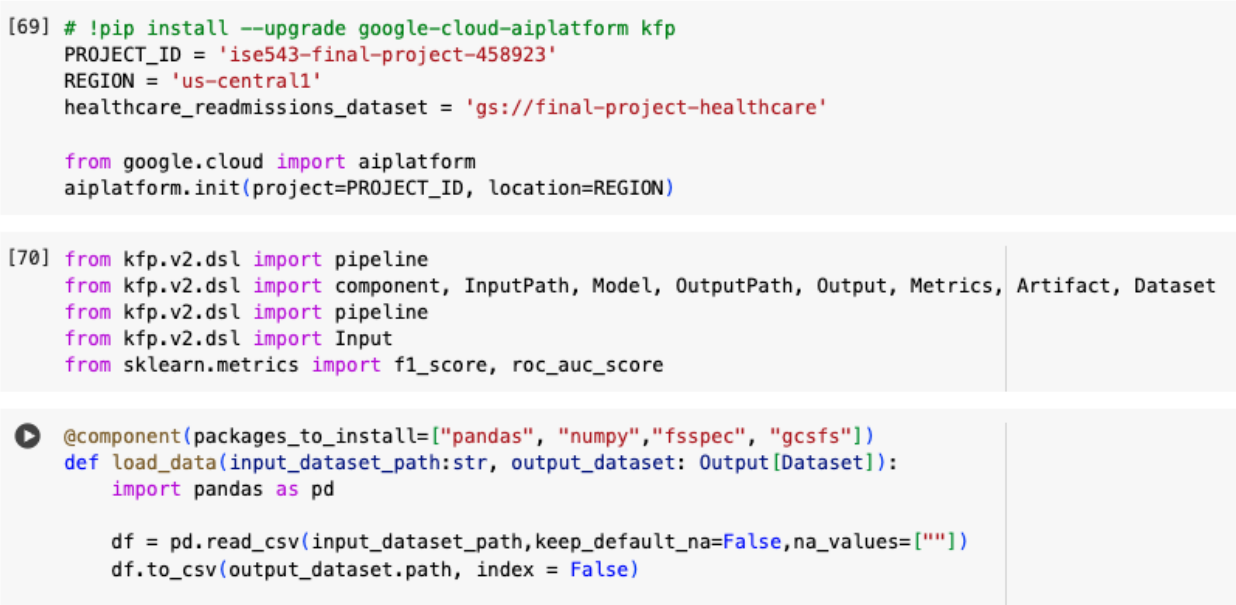
TRAINING PIPELINE





TRAINING PIPELINE

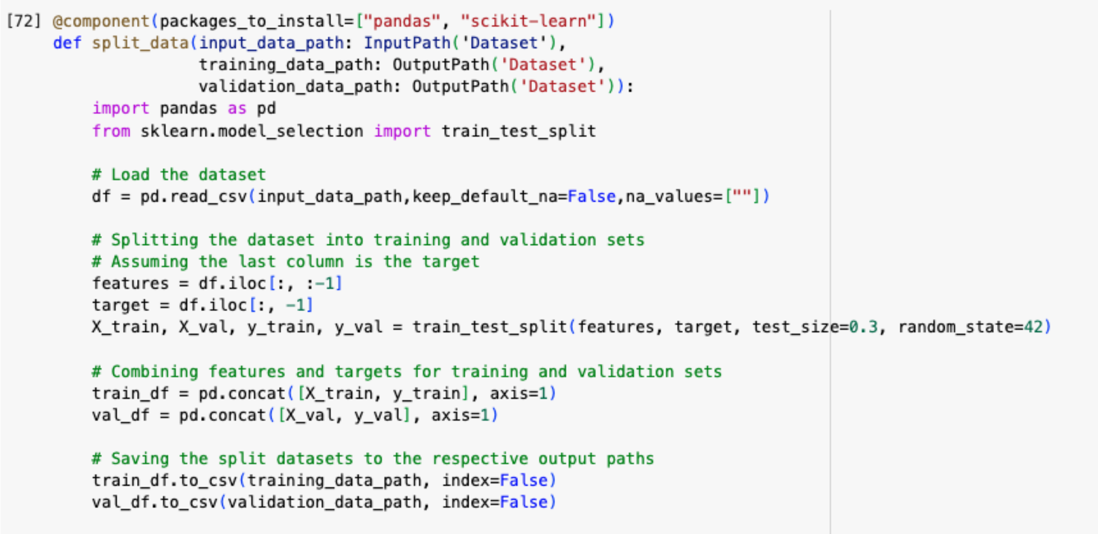
PIPELINE DEFINITION CODE



```
[69] # !pip install --upgrade google-cloud-aiplatform kfp
     PROJECT_ID = 'ise543-final-project-458923'
     REGION = 'us-central1'
     healthcare_readmissions_dataset = 'gs://final-project-healthcare'

     from google.cloud import aiplatform
     aiplatform.init(project=PROJECT_ID, location=REGION)
```

```
[70] from kfp.v2.dsl import pipeline
     from kfp.v2.dsl import component, InputPath, Model, OutputPath, Output, Metrics, Artifact, Dataset
     from kfp.v2.dsl import pipeline
     from kfp.v2.dsl import Input
     from sklearn.metrics import f1_score, roc_auc_score
```

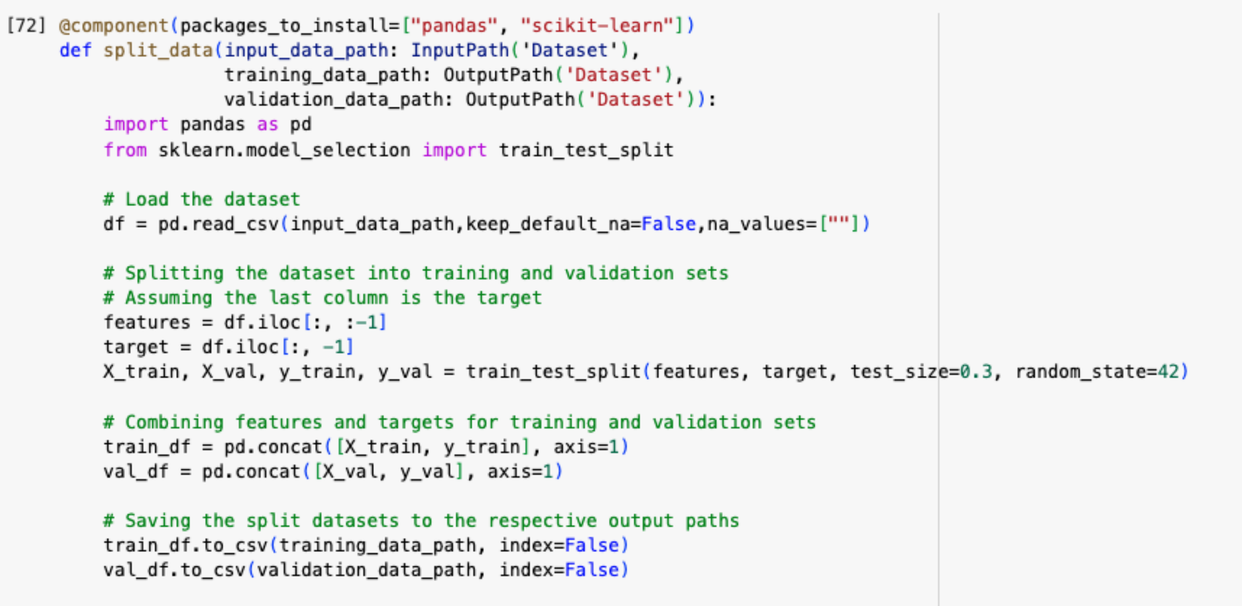
```
@component(packages_to_install=["pandas", "numpy","fsspec", "gcsfs"])
def load_data(input_dataset_path:str, output_dataset: Output[Dataset]):
    import pandas as pd

    df = pd.read_csv(input_dataset_path,keep_default_na=False,na_values=[""])
    df.to_csv(output_dataset.path, index = False)
```



```
[72] @component(packages_to_install=["pandas", "scikit-learn"])
     def split_data(input_data_path: InputPath('Dataset'),
                    training_data_path: OutputPath('Dataset'),
                    validation_data_path: OutputPath('Dataset')):
         import pandas as pd
         from sklearn.model_selection import train_test_split

         # Load the dataset
         df = pd.read_csv(input_data_path,keep_default_na=False,na_values=[""])

         # Splitting the dataset into training and validation sets
         # Assuming the last column is the target
         features = df.iloc[:, :-1]
         target = df.iloc[:, -1]
         X_train, X_val, y_train, y_val = train_test_split(features, target, test_size=0.3, random_state=42)

         # Combining features and targets for training and validation sets
         train_df = pd.concat([X_train, y_train], axis=1)
         val_df = pd.concat([X_val, y_val], axis=1)

         # Saving the split datasets to the respective output paths
         train_df.to_csv(training_data_path, index=False)
         val_df.to_csv(validation_data_path, index=False)
```

TRAINING PIPELINE

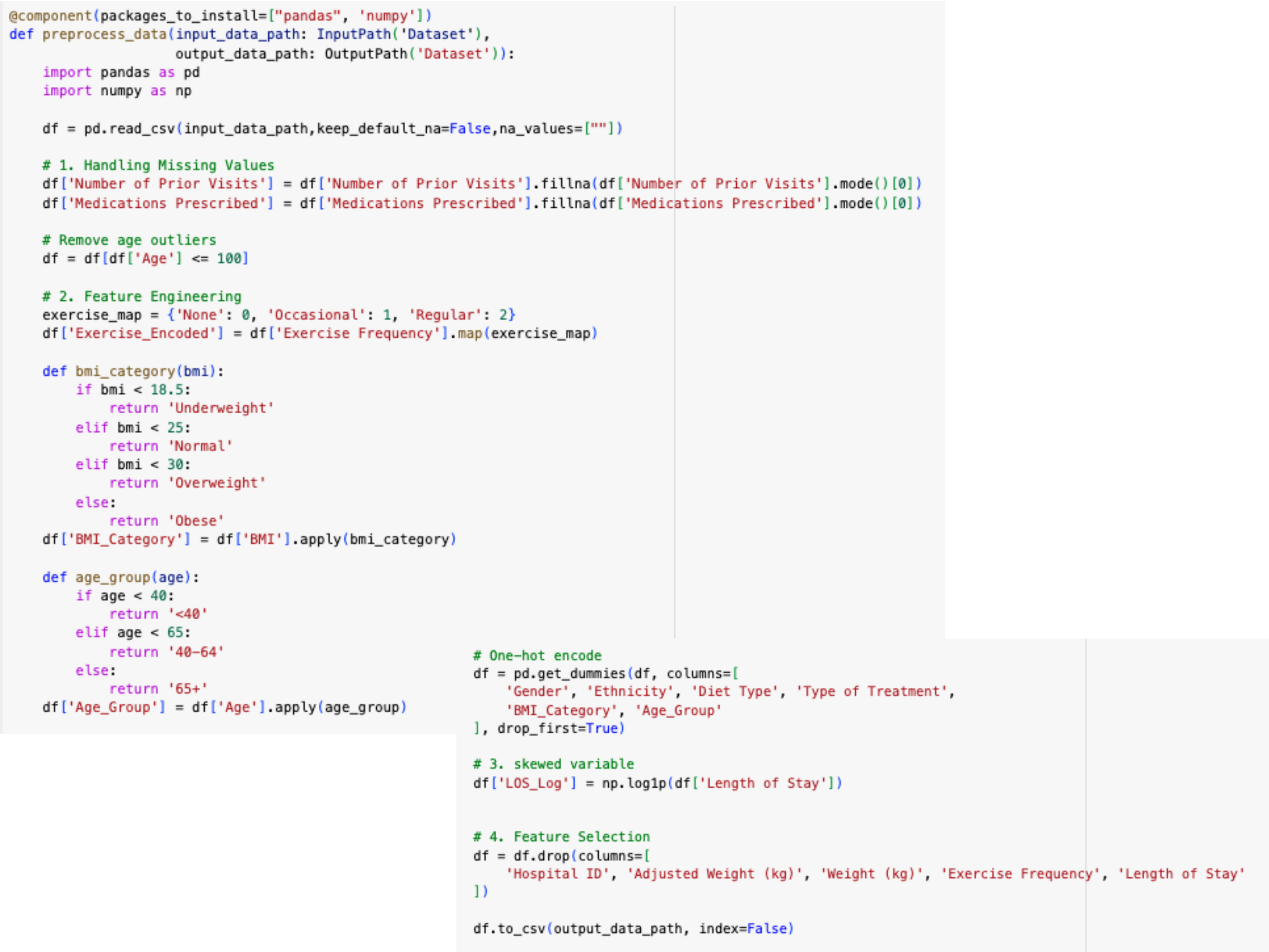
PIPELINE DEFINITION CODE



```
[72] @component(packages_to_install=["pandas", "scikit-learn"])
     def split_data(input_data_path: InputPath('Dataset'),
                    training_data_path: OutputPath('Dataset'),
                    validation_data_path: OutputPath('Dataset')):
         import pandas as pd
         from sklearn.model_selection import train_test_split

         # Load the dataset
         df = pd.read_csv(input_data_path,keep_default_na=False,na_values=[""])

         # Splitting the dataset into training and validation sets
         # Assuming the last column is the target
         features = df.iloc[:, :-1]
         target = df.iloc[:, -1]
         X_train, X_val, y_train, y_val = train_test_split(features, target, test_size=0.3, random_state=42)

         # Combining features and targets for training and validation sets
         train_df = pd.concat([X_train, y_train], axis=1)
         val_df = pd.concat([X_val, y_val], axis=1)

         # Saving the split datasets to the respective output paths
         train_df.to_csv(training_data_path, index=False)
         val_df.to_csv(validation_data_path, index=False)
```

TRAINING PIPELINE

PIPELINE DEFINITION CODE

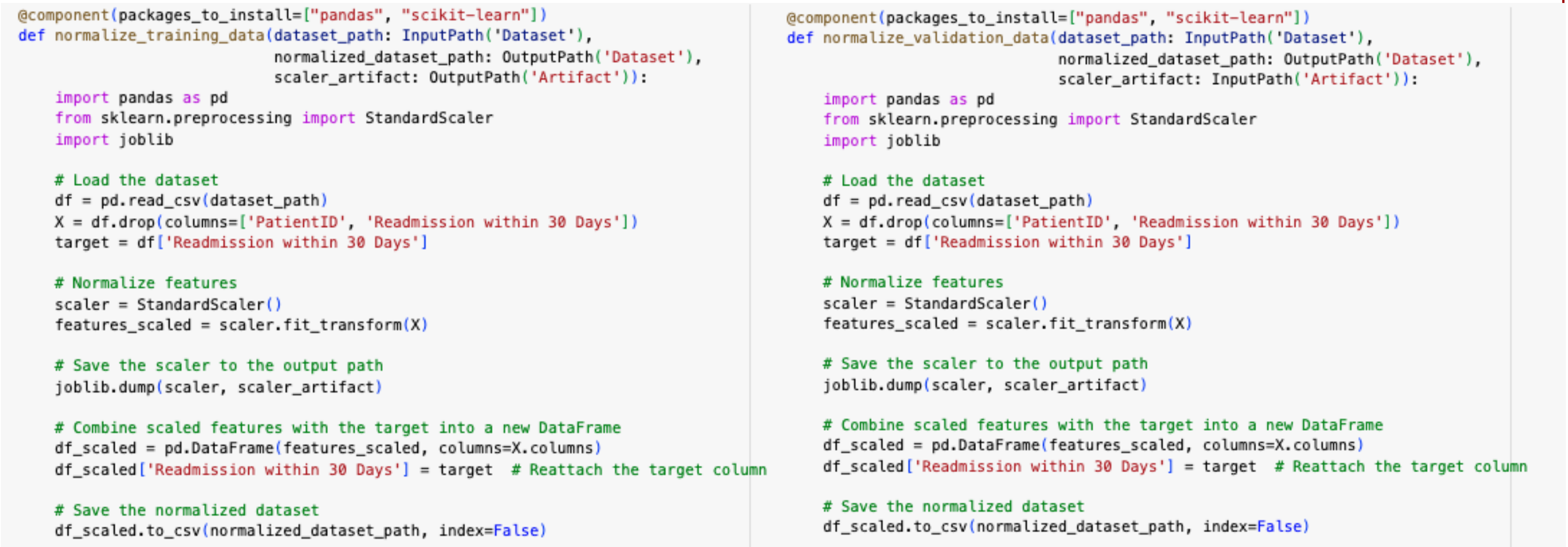


```
@component(packages_to_install=["pandas", 'numpy'])
def preprocess_data(input_data_path: InputPath('Dataset'),
                    output_data_path: OutputPath('Dataset')):
    import pandas as pd
    import numpy as np

    df = pd.read_csv(input_data_path,keep_default_na=False,na_values=[""])

    # 1. Handling Missing Values
    df['Number of Prior Visits'] = df['Number of Prior Visits'].fillna(df['Number of Prior Visits'].mode()[0])
    df['Medications Prescribed'] = df['Medications Prescribed'].fillna(df['Medications Prescribed'].mode()[0])

    # Remove age outliers
    df = df[df['Age'] <= 100]

    # 2. Feature Engineering
    exercise_map = {'None': 0, 'Occasional': 1, 'Regular': 2}
    df['Exercise_Encoded'] = df['Exercise Frequency'].map(exercise_map)

    def bmi_category(bmi):
        if bmi < 18.5:
            return 'Underweight'
        elif bmi < 25:
            return 'Normal'
        elif bmi < 30:
            return 'Overweight'
        else:
            return 'Obese'
    df['BMI_Category'] = df['BMI'].apply(bmi_category)

    def age_group(age):
        if age < 40:
            return '<40'
        elif age < 65:
            return '40-64'
        else:
            return '65+'
    df['Age_Group'] = df['Age'].apply(age_group)
```

```
# One-hot encode
df = pd.get_dummies(df, columns=[
    'Gender', 'Ethnicity', 'Diet Type', 'Type of Treatment',
    'BMI_Category', 'Age_Group'
], drop_first=True)

# 3. skewed variable
df['LOS_Log'] = np.log1p(df['Length of Stay'])


# 4. Feature Selection
df = df.drop(columns=[
    'Hospital ID', 'Adjusted Weight (kg)', 'Weight (kg)', 'Exercise Frequency', 'Length of Stay'
])

df.to_csv(output_data_path, index=False)
```

TRAINING PIPELINE

PIPELINE DEFINITION CODE

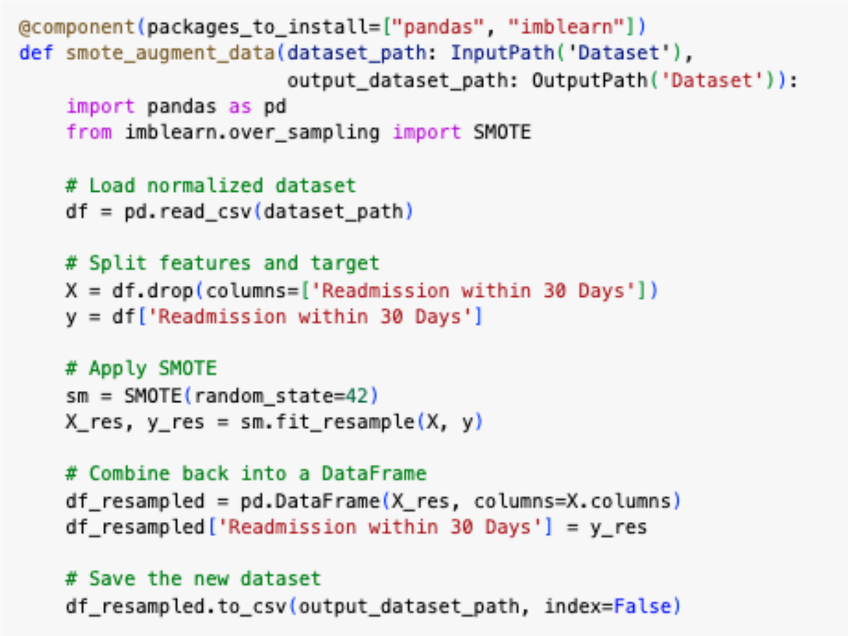


```
@component(packages_to_install=["pandas", "scikit-learn"])
def normalize_training_data(dataset_path: InputPath('Dataset'),
                            normalized_dataset_path: OutputPath('Dataset'),
                            scaler_artifact: OutputPath('Artifact')):
    import pandas as pd
    from sklearn.preprocessing import StandardScaler
    import joblib

    # Load the dataset
    df = pd.read_csv(dataset_path)
    X = df.drop(columns=['PatientID', 'Readmission within 30 Days'])
    target = df['Readmission within 30 Days']

    # Normalize features
    scaler = StandardScaler()
    features_scaled = scaler.fit_transform(X)

    # Save the scaler to the output path
    joblib.dump(scaler, scaler_artifact)

    # Combine scaled features with the target into a new DataFrame
    df_scaled = pd.DataFrame(features_scaled, columns=X.columns)
    df_scaled['Readmission within 30 Days'] = target  # Reattach the target column

    # Save the normalized dataset
    df_scaled.to_csv(normalized_dataset_path, index=False)
```

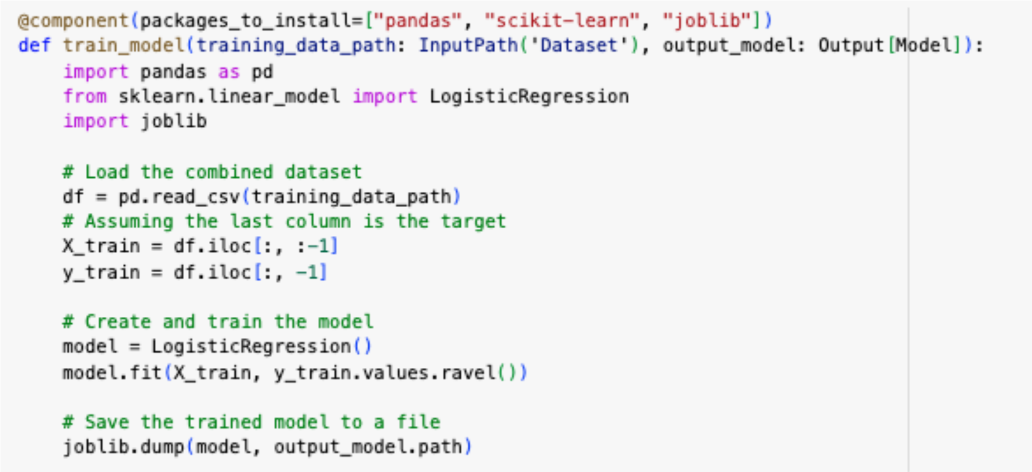
```
@component(packages_to_install=["pandas", "scikit-learn"])
def normalize_validation_data(dataset_path: InputPath('Dataset'),
                              normalized_dataset_path: OutputPath('Dataset'),
                              scaler_artifact: InputPath('Artifact')):
    import pandas as pd
    from sklearn.preprocessing import StandardScaler
    import joblib

    # Load the dataset
    df = pd.read_csv(dataset_path)
    X = df.drop(columns=['PatientID', 'Readmission within 30 Days'])
    target = df['Readmission within 30 Days']

    # Normalize features
    scaler = StandardScaler()
    features_scaled = scaler.fit_transform(X)

    # Save the scaler to the output path
    joblib.dump(scaler, scaler_artifact)

    # Combine scaled features with the target into a new DataFrame
    df_scaled = pd.DataFrame(features_scaled, columns=X.columns)
    df_scaled['Readmission within 30 Days'] = target  # Reattach the target column

    # Save the normalized dataset
    df_scaled.to_csv(normalized_dataset_path, index=False)
```

TRAINING PIPELINE

PIPELINE DEFINITION CODE



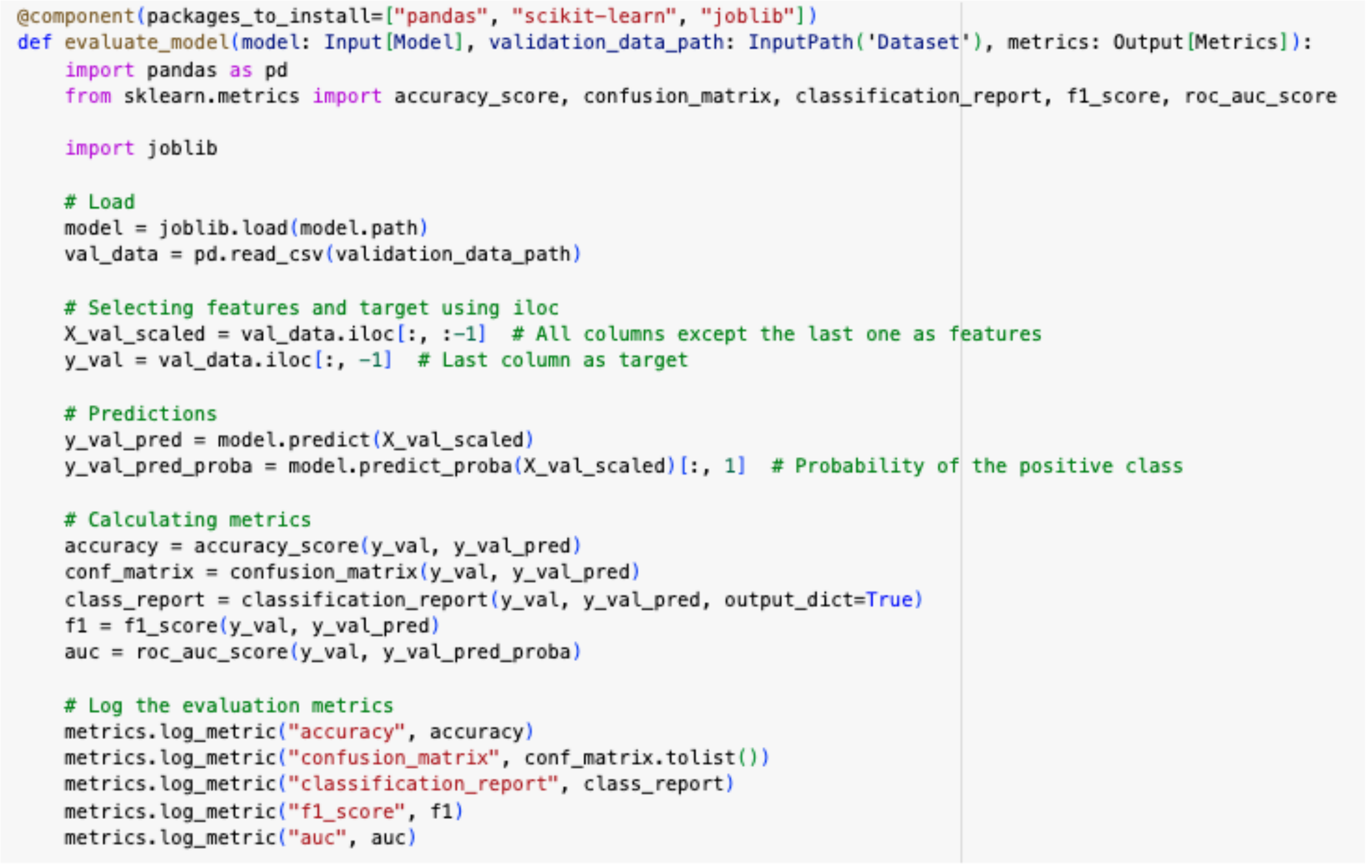
```
@component(packages_to_install=["pandas", "imblearn"])
def smote_augment_data(dataset_path: InputPath('Dataset'),
                       output_dataset_path: OutputPath('Dataset')):
    import pandas as pd
    from imblearn.over_sampling import SMOTE

    # Load normalized dataset
    df = pd.read_csv(dataset_path)

    # Split features and target
    X = df.drop(columns=['Readmission within 30 Days'])
    y = df['Readmission within 30 Days']

    # Apply SMOTE
    sm = SMOTE(random_state=42)
    X_res, y_res = sm.fit_resample(X, y)

    # Combine back into a DataFrame
    df_resampled = pd.DataFrame(X_res, columns=X.columns)
    df_resampled['Readmission within 30 Days'] = y_res

    # Save the new dataset
    df_resampled.to_csv(output_dataset_path, index=False)
```



```
@component(packages_to_install=["pandas", "scikit-learn", "joblib"])
def train_model(training_data_path: InputPath('Dataset'), output_model: Output[Model]):
    import pandas as pd
    from sklearn.linear_model import LogisticRegression
    import joblib

    # Load the combined dataset
    df = pd.read_csv(training_data_path)
    # Assuming the last column is the target
    X_train = df.iloc[:, :-1]
    y_train = df.iloc[:, -1]

    # Create and train the model
    model = LogisticRegression()
    model.fit(X_train, y_train.values.ravel())

    # Save the trained model to a file
    joblib.dump(model, output_model.path)
```

TRAINING PIPELINE

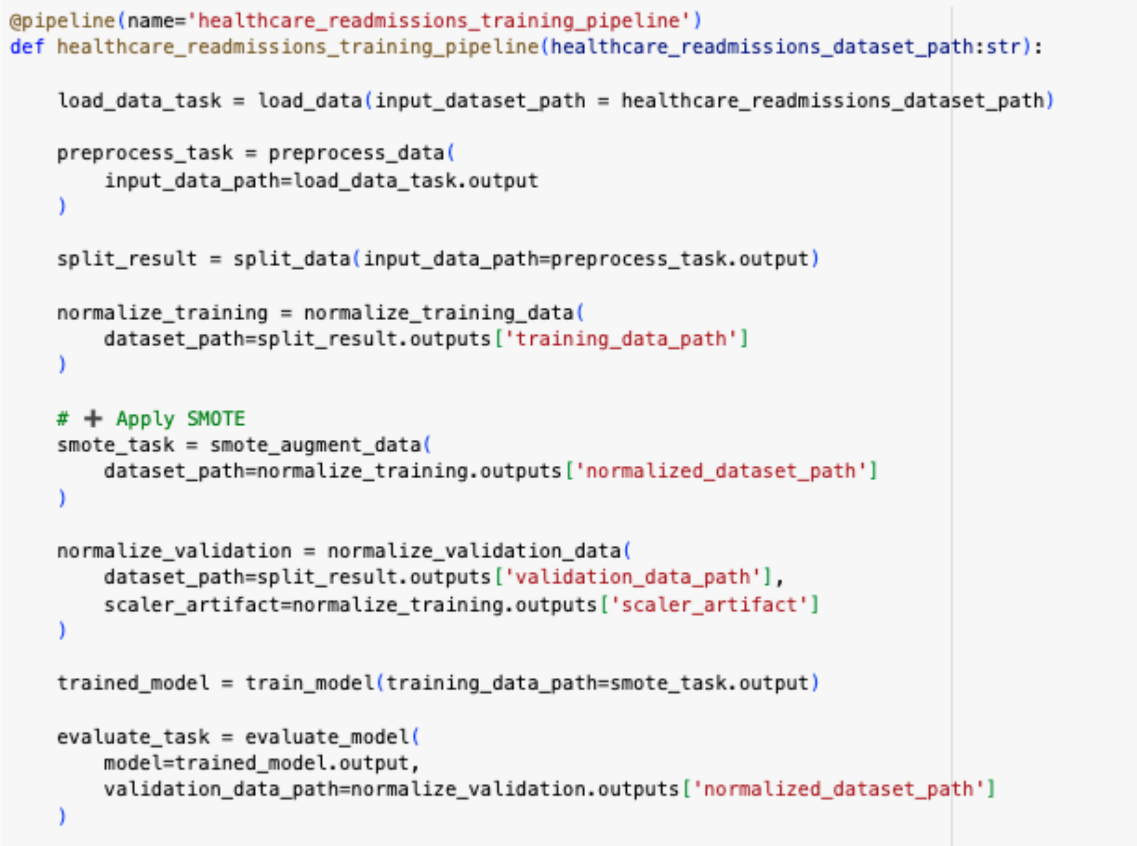
PIPELINE DEFINITION CODE



```python
@component(packages_to_install=["pandas", "scikit-learn", "joblib"])
def evaluate_model(model: Input[Model], validation_data_path: InputPath('Dataset'), metrics: Output[Metrics]):
    import pandas as pd
    from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, f1_score, roc_auc_score

    import joblib

    # Load
    model = joblib.load(model.path)
    val_data = pd.read_csv(validation_data_path)

    # Selecting features and target using iloc
    X_val_scaled = val_data.iloc[:, :-1]  # All columns except the last one as features
    y_val = val_data.iloc[:, -1]  # Last column as target

    # Predictions
    y_val_pred = model.predict(X_val_scaled)
    y_val_pred_proba = model.predict_proba(X_val_scaled)[:, 1]  # Probability of the positive class

    # Calculating metrics
    accuracy = accuracy_score(y_val, y_val_pred)
    conf_matrix = confusion_matrix(y_val, y_val_pred)
    class_report = classification_report(y_val, y_val_pred, output_dict=True)
    f1 = f1_score(y_val, y_val_pred)
    auc = roc_auc_score(y_val, y_val_pred_proba)

    # Log the evaluation metrics
    metrics.log_metric("accuracy", accuracy)
    metrics.log_metric("confusion_matrix", conf_matrix.tolist())
    metrics.log_metric("classification_report", class_report)
    metrics.log_metric("f1_score", f1)
    metrics.log_metric("auc", auc)
```

TRAINING PIPELINE

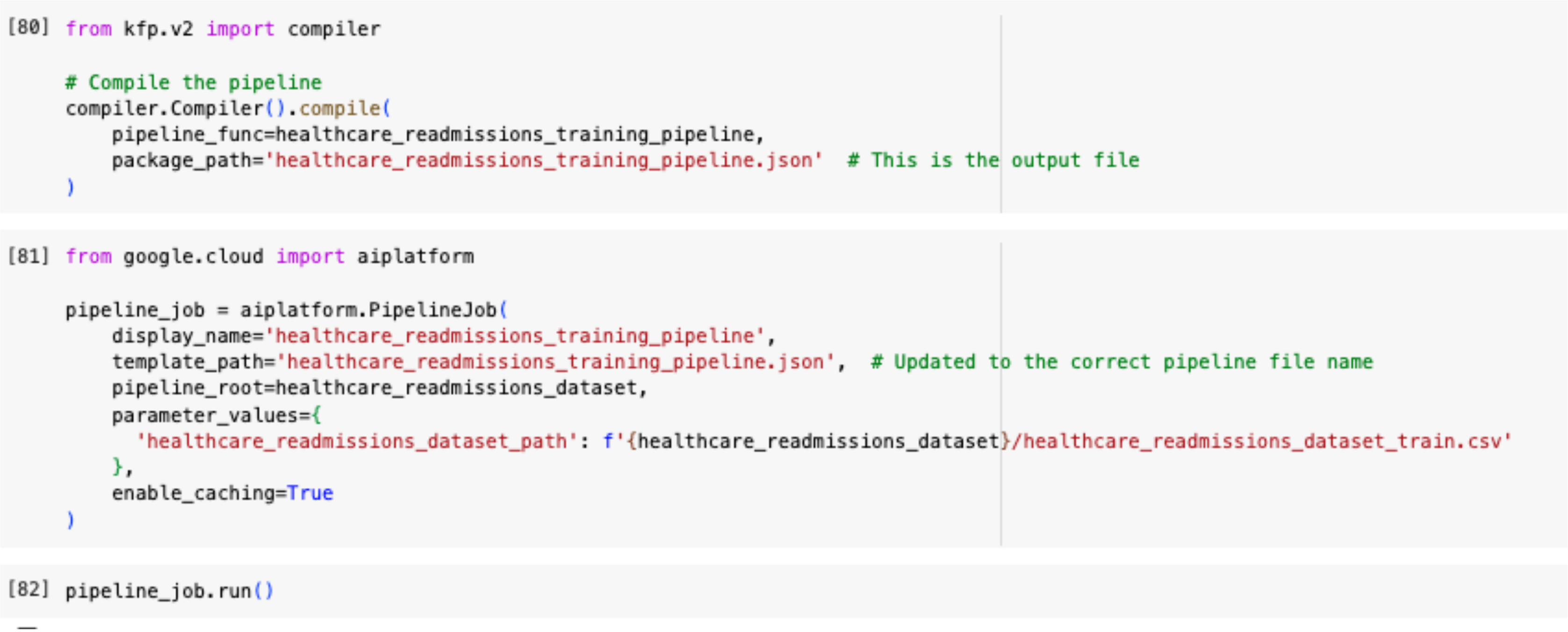
PIPELINE DEFINITION CODE



```python
@pipeline(name='healthcare_readmissions_training_pipeline')
def healthcare_readmissions_training_pipeline(healthcare_readmissions_dataset_path:str):

    load_data_task = load_data(input_dataset_path = healthcare_readmissions_dataset_path)

    preprocess_task = preprocess_data(
        input_data_path=load_data_task.output
    )

    split_result = split_data(input_data_path=preprocess_task.output)

    normalize_training = normalize_training_data(
        dataset_path=split_result.outputs['training_data_path']
    )

    # ➕ Apply SMOTE
    smote_task = smote_augment_data(
        dataset_path=normalize_training.outputs['normalized_dataset_path']
    )

    normalize_validation = normalize_validation_data(
        dataset_path=split_result.outputs['validation_data_path'],
        scaler_artifact=normalize_training.outputs['scaler_artifact']
    )

    trained_model = train_model(training_data_path=smote_task.output)

    evaluate_task = evaluate_model(
        model=trained_model.output,
        validation_data_path=normalize_validation.outputs['normalized_dataset_path']
    )
```

TRAINING PIPELINE

PIPELINE DEFINITION CODE



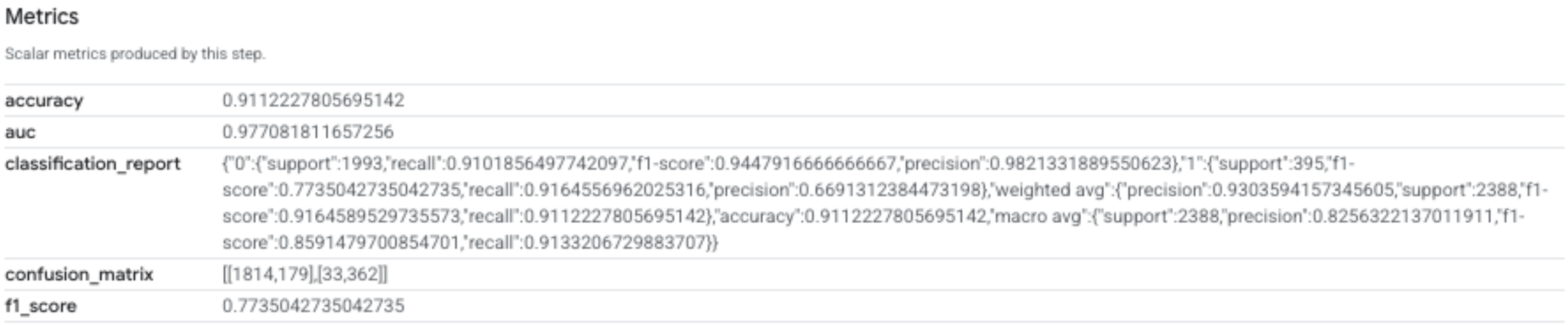
```
[80]  from kfp.v2 import compiler

      # Compile the pipeline
      compiler.Compiler().compile(
          pipeline_func=healthcare_readmissions_training_pipeline,
          package_path='healthcare_readmissions_training_pipeline.json'  # This is the output file
      )
```

```
[81]  from google.cloud import aiplatform

      pipeline_job = aiplatform.PipelineJob(
          display_name='healthcare_readmissions_training_pipeline',
          template_path='healthcare_readmissions_training_pipeline.json',  # Updated to the correct pipeline file name
          pipeline_root=healthcare_readmissions_dataset,
          parameter_values={
            'healthcare_readmissions_dataset_path': f'{healthcare_readmissions_dataset}/healthcare_readmissions_dataset_train.csv'
          },
          enable_caching=True
      )
```

```
[82]  pipeline_job.run()
```

TRAINING PIPELINE

MODEL ASSESSMENT STATISTICS



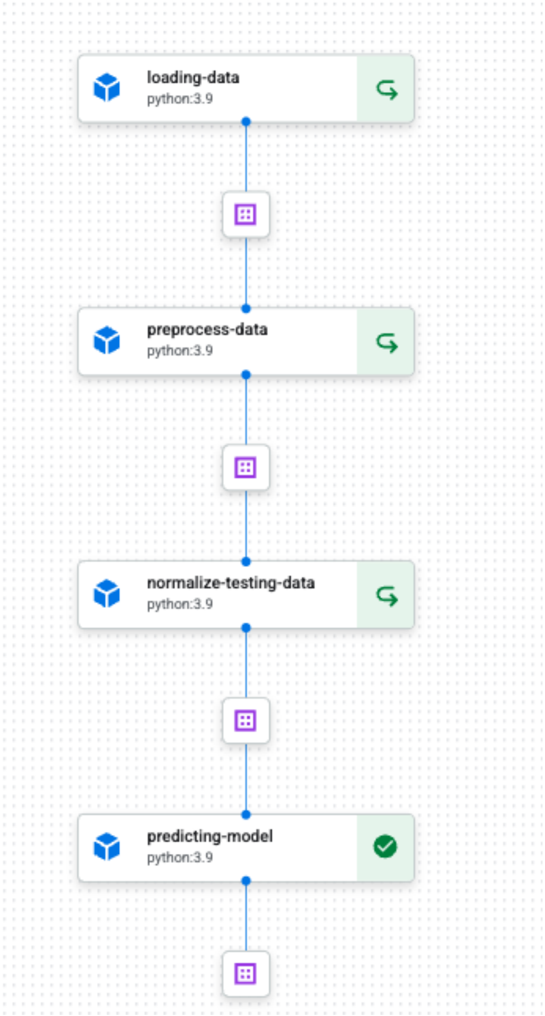
Metrics

Scalar metrics produced by this step.

accuracy	0.9112227805695142
auc	0.977081811657256
classification_report	{"0":{"support":1993,"recall":0.9101856497742097,"f1-score":0.9447916666666667,"precision":0.9821331889550623},"1":{"support":395,"f1-score":0.7735042735042735,"recall":0.9164556962025316,"precision":0.6691312384473198},"weighted avg":{"precision":0.9303594157345605,"support":2388,"f1-score":0.9164589529735573,"recall":0.9112227805695142},"accuracy":0.9112227805695142,"macro avg":{"support":2388,"precision":0.8256322137011911,"f1-score":0.8591479700854701,"recall":0.9133206729883707}}
confusion_matrix	[[1814,179],[33,362]]
f1_score	0.7735042735042735

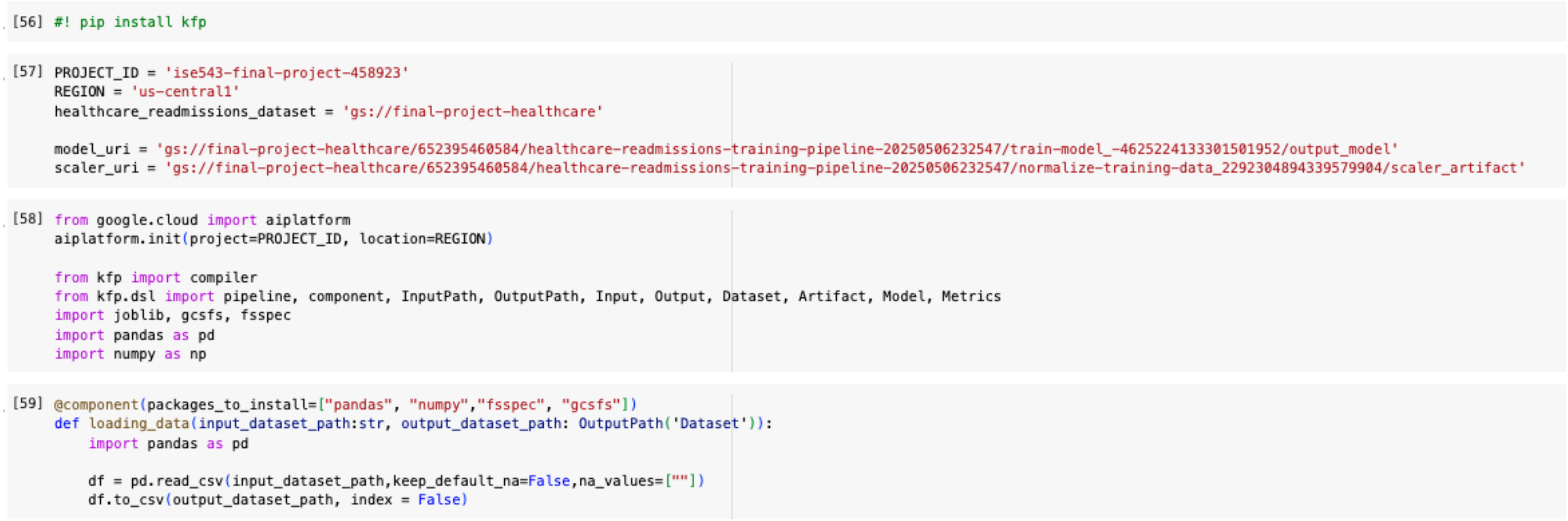
INFERENCE PIPELINE

DEFINE INFERENCE PIPELINE



INFERENCE PIPELINE

DEFINE INFERENCE PIPELINE



```
[56] #! pip install kfp
```

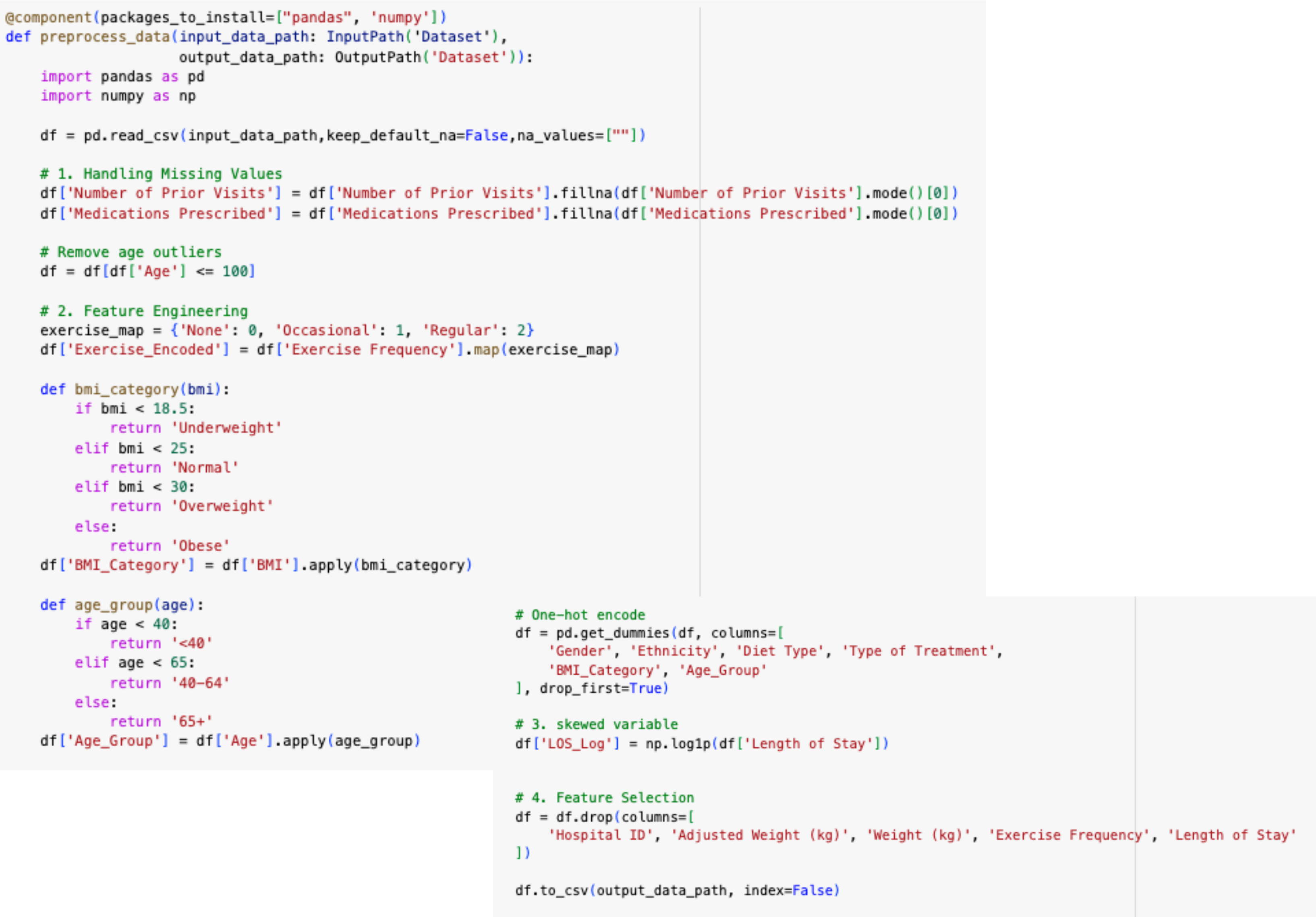
```
[57] PROJECT_ID = 'ise543-final-project-458923'
     REGION = 'us-central1'
     healthcare_readmissions_dataset = 'gs://final-project-healthcare'

     model_uri = 'gs://final-project-healthcare/652395460584/healthcare-readmissions-training-pipeline-20250506232547/train-model_-4625224133301501952/output_model'
     scaler_uri = 'gs://final-project-healthcare/652395460584/healthcare-readmissions-training-pipeline-20250506232547/normalize-training-data_2292304894339579904/scaler_artifact'
```

```
[58] from google.cloud import aiplatform
     aiplatform.init(project=PROJECT_ID, location=REGION)

     from kfp import compiler
     from kfp.dsl import pipeline, component, InputPath, OutputPath, Input, Output, Dataset, Artifact, Model, Metrics
     import joblib, gcsfs, fsspec
     import pandas as pd
     import numpy as np
```

```
[59] @component(packages_to_install=["pandas", "numpy","fsspec", "gcsfs"])
     def loading_data(input_dataset_path:str, output_dataset_path: OutputPath('Dataset')):
         import pandas as pd

         df = pd.read_csv(input_dataset_path,keep_default_na=False,na_values=[""])
         df.to_csv(output_dataset_path, index = False)
```

INFERENCE PIPELINE

DEFINE INFERENCE PIPELINE

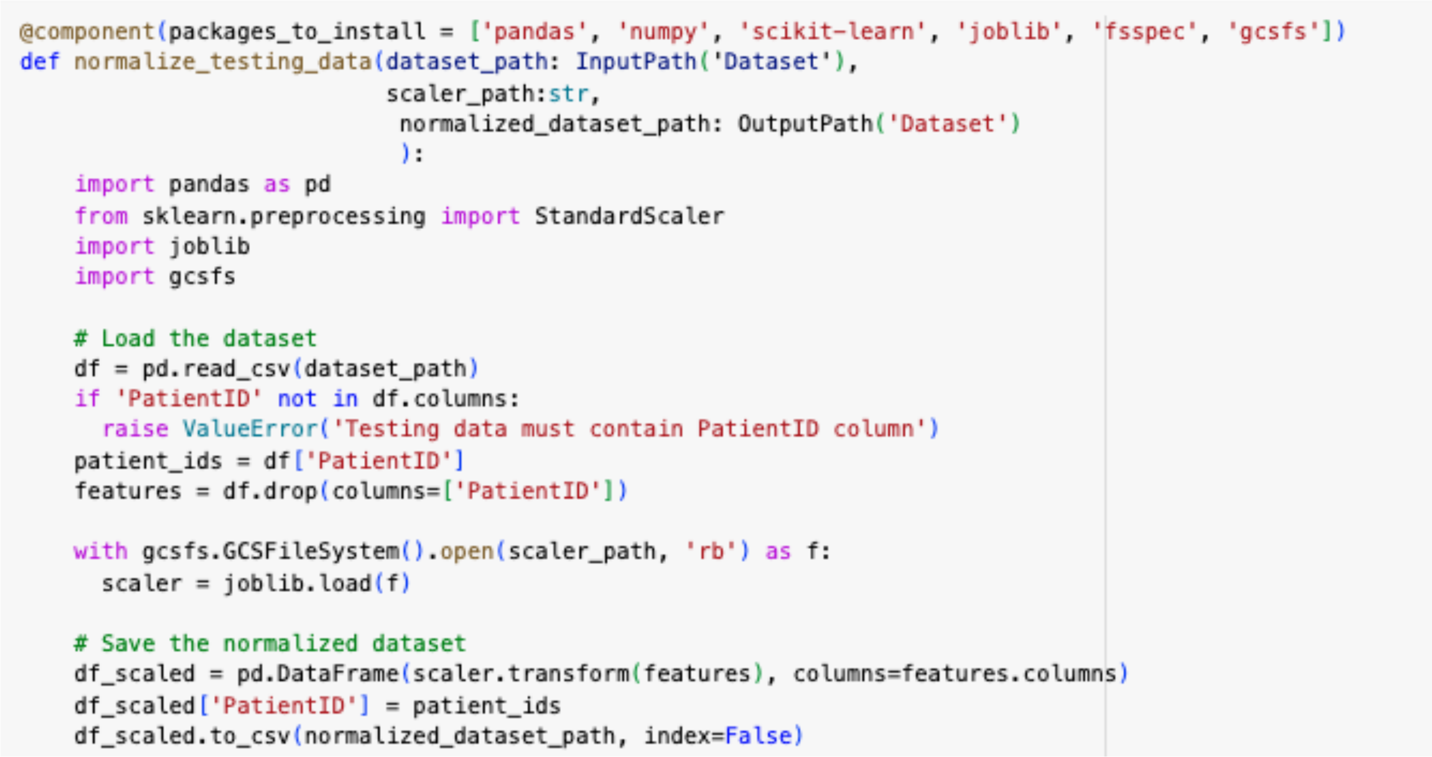


```
@component(packages_to_install=["pandas", 'numpy'])
def preprocess_data(input_data_path: InputPath('Dataset'),
                    output_data_path: OutputPath('Dataset')):
    import pandas as pd
    import numpy as np

    df = pd.read_csv(input_data_path,keep_default_na=False,na_values=[""])

    # 1. Handling Missing Values
    df['Number of Prior Visits'] = df['Number of Prior Visits'].fillna(df['Number of Prior Visits'].mode()[0])
    df['Medications Prescribed'] = df['Medications Prescribed'].fillna(df['Medications Prescribed'].mode()[0])

    # Remove age outliers
    df = df[df['Age'] <= 100]

    # 2. Feature Engineering
    exercise_map = {'None': 0, 'Occasional': 1, 'Regular': 2}
    df['Exercise_Encoded'] = df['Exercise Frequency'].map(exercise_map)

    def bmi_category(bmi):
        if bmi < 18.5:
            return 'Underweight'
        elif bmi < 25:
            return 'Normal'
        elif bmi < 30:
            return 'Overweight'
        else:
            return 'Obese'
    df['BMI_Category'] = df['BMI'].apply(bmi_category)

    def age_group(age):
        if age < 40:
            return '<40'
        elif age < 65:
            return '40-64'
        else:
            return '65+'
    df['Age_Group'] = df['Age'].apply(age_group)
```

```
# One-hot encode
df = pd.get_dummies(df, columns=[
    'Gender', 'Ethnicity', 'Diet Type', 'Type of Treatment',
    'BMI_Category', 'Age_Group'
], drop_first=True)

# 3. skewed variable
df['LOS_Log'] = np.log1p(df['Length of Stay'])

# 4. Feature Selection
df = df.drop(columns=[
    'Hospital ID', 'Adjusted Weight (kg)', 'Weight (kg)', 'Exercise Frequency', 'Length of Stay'
])

df.to_csv(output_data_path, index=False)
```

INFERENCE PIPELINE

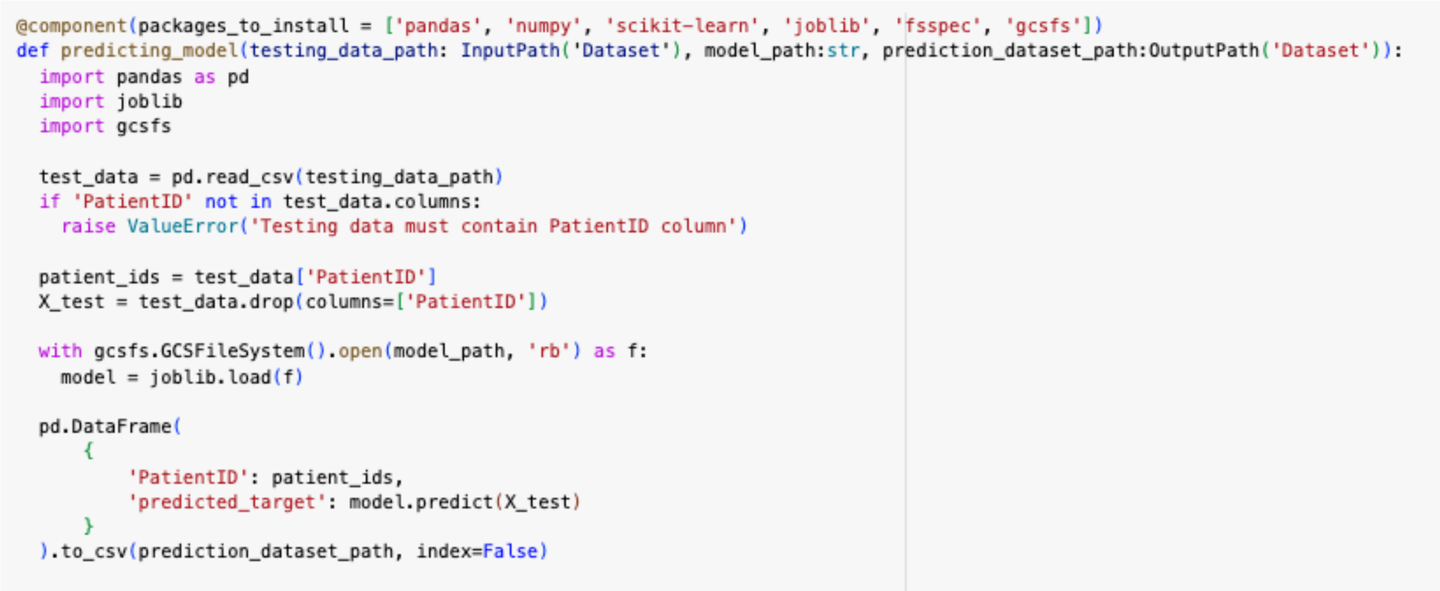
DEFINE INFERENCE PIPELINE



```
@component(packages_to_install = ['pandas', 'numpy', 'scikit-learn', 'joblib', 'fsspec', 'gcsfs'])
def normalize_testing_data(dataset_path: InputPath('Dataset'),
                           scaler_path:str,
                            normalized_dataset_path: OutputPath('Dataset')
                            ):
    import pandas as pd
    from sklearn.preprocessing import StandardScaler
    import joblib
    import gcsfs

    # Load the dataset
    df = pd.read_csv(dataset_path)
    if 'PatientID' not in df.columns:
      raise ValueError('Testing data must contain PatientID column')
    patient_ids = df['PatientID']
    features = df.drop(columns=['PatientID'])

    with gcsfs.GCSFileSystem().open(scaler_path, 'rb') as f:
      scaler = joblib.load(f)

    # Save the normalized dataset
    df_scaled = pd.DataFrame(scaler.transform(features), columns=features.columns)
    df_scaled['PatientID'] = patient_ids
    df_scaled.to_csv(normalized_dataset_path, index=False)
```

INFERENCE PIPELINE

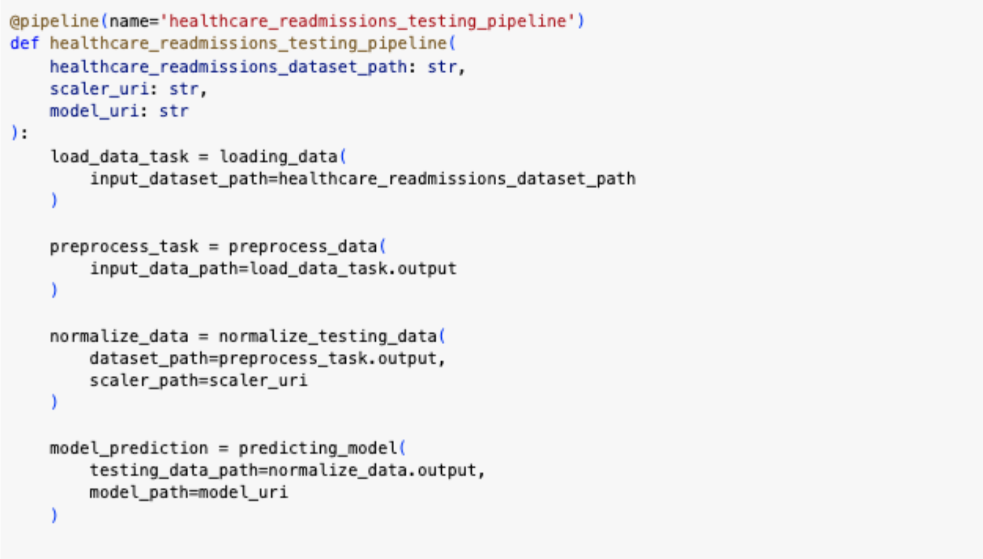
DEFINE INFERENCE PIPELINE



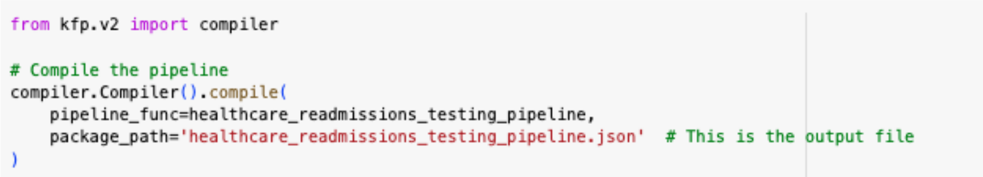
```
@component(packages_to_install = ['pandas', 'numpy', 'scikit-learn', 'joblib', 'fsspec', 'gcsfs'])
def predicting_model(testing_data_path: InputPath('Dataset'), model_path:str, prediction_dataset_path:OutputPath('Dataset')):
  import pandas as pd
  import joblib
  import gcsfs

  test_data = pd.read_csv(testing_data_path)
  if 'PatientID' not in test_data.columns:
    raise ValueError('Testing data must contain PatientID column')

  patient_ids = test_data['PatientID']
  X_test = test_data.drop(columns=['PatientID'])

  with gcsfs.GCSFileSystem().open(model_path, 'rb') as f:
    model = joblib.load(f)

  pd.DataFrame(
      {
          'PatientID': patient_ids,
          'predicted_target': model.predict(X_test)
      }
  ).to_csv(prediction_dataset_path, index=False)
```

INFERENCE PIPELINE

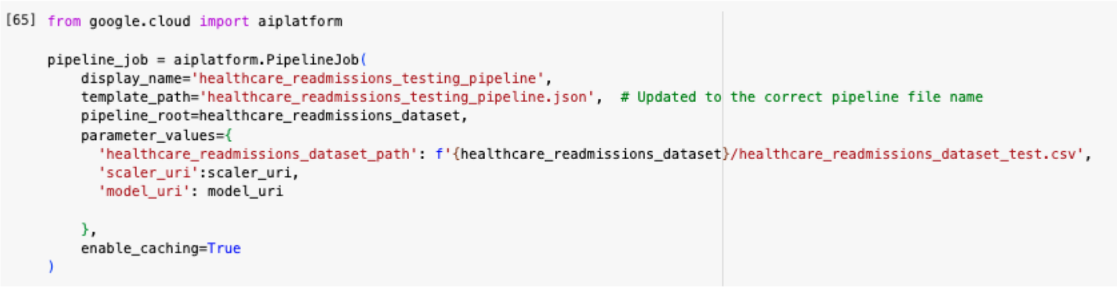
DEFINE INFERENCE PIPELINE



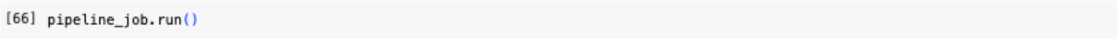
```
@pipeline(name='healthcare_readmissions_testing_pipeline')
def healthcare_readmissions_testing_pipeline(
    healthcare_readmissions_dataset_path: str,
    scaler_uri: str,
    model_uri: str
):
    load_data_task = loading_data(
        input_dataset_path=healthcare_readmissions_dataset_path
    )

    preprocess_task = preprocess_data(
        input_data_path=load_data_task.output
    )

    normalize_data = normalize_testing_data(
        dataset_path=preprocess_task.output,
        scaler_path=scaler_uri
    )

    model_prediction = predicting_model(
        testing_data_path=normalize_data.output,
        model_path=model_uri
    )
```



```
from kfp.v2 import compiler

# Compile the pipeline
compiler.Compiler().compile(
    pipeline_func=healthcare_readmissions_testing_pipeline,
    package_path='healthcare_readmissions_testing_pipeline.json'  # This is the output file
)
```



```
[65] from google.cloud import aiplatform

     pipeline_job = aiplatform.PipelineJob(
         display_name='healthcare_readmissions_testing_pipeline',
         template_path='healthcare_readmissions_testing_pipeline.json',  # Updated to the correct pipeline file name
         pipeline_root=healthcare_readmissions_dataset,
         parameter_values={
           'healthcare_readmissions_dataset_path': f'{healthcare_readmissions_dataset}/healthcare_readmissions_dataset_test.csv',
           'scaler_uri':scaler_uri,
           'model_uri': model_uri

         },
         enable_caching=True
     )
```



```
[66] pipeline_job.run()
```

SUMMARY DISCUSSION

- The logistic regression model appears to not be a good fit for this dataset
  - Next steps would be to try other model types and an ensemble model
- Do more research on healthcare domain knowledge to do better variable selection and appropriate feature engineering