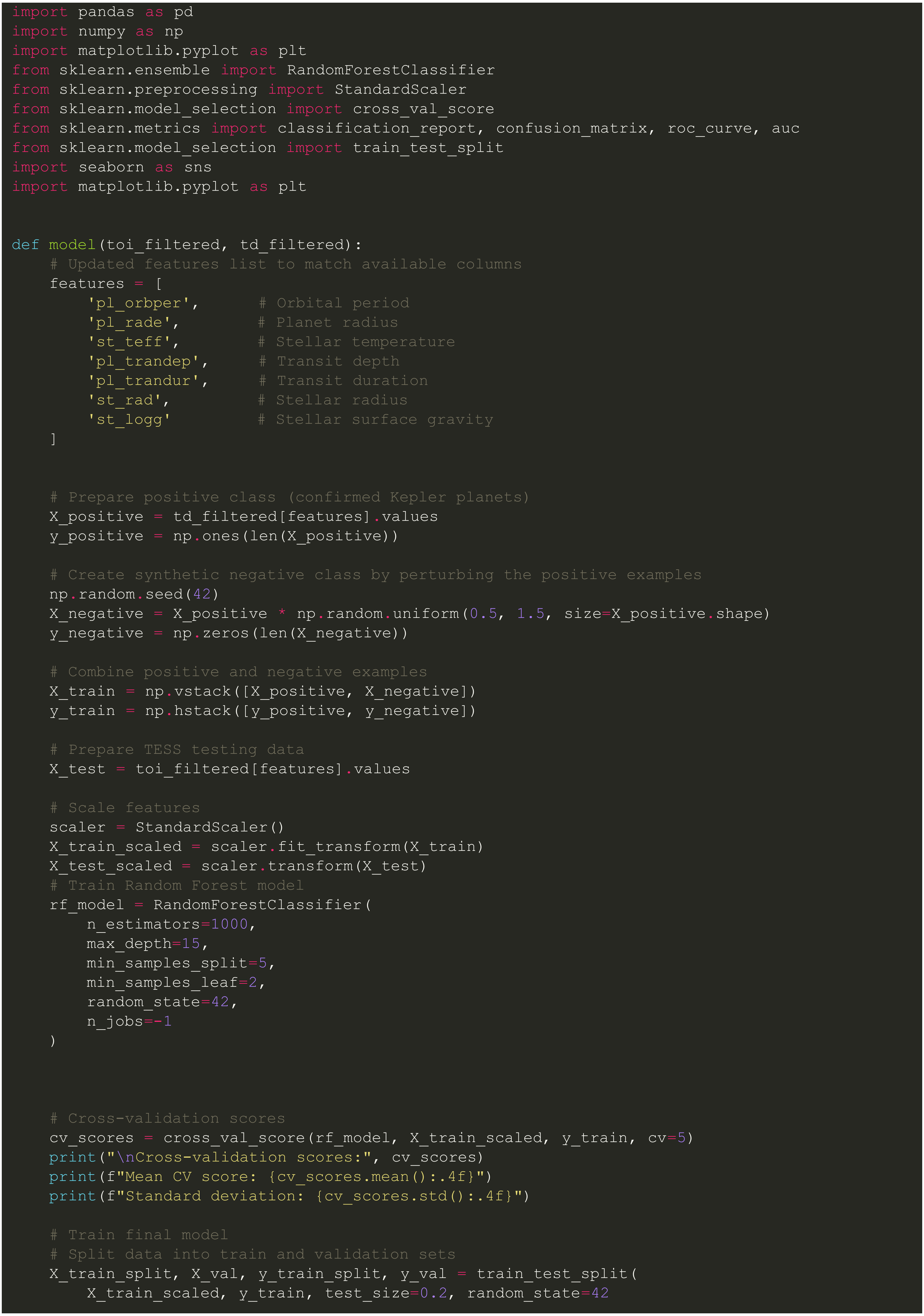
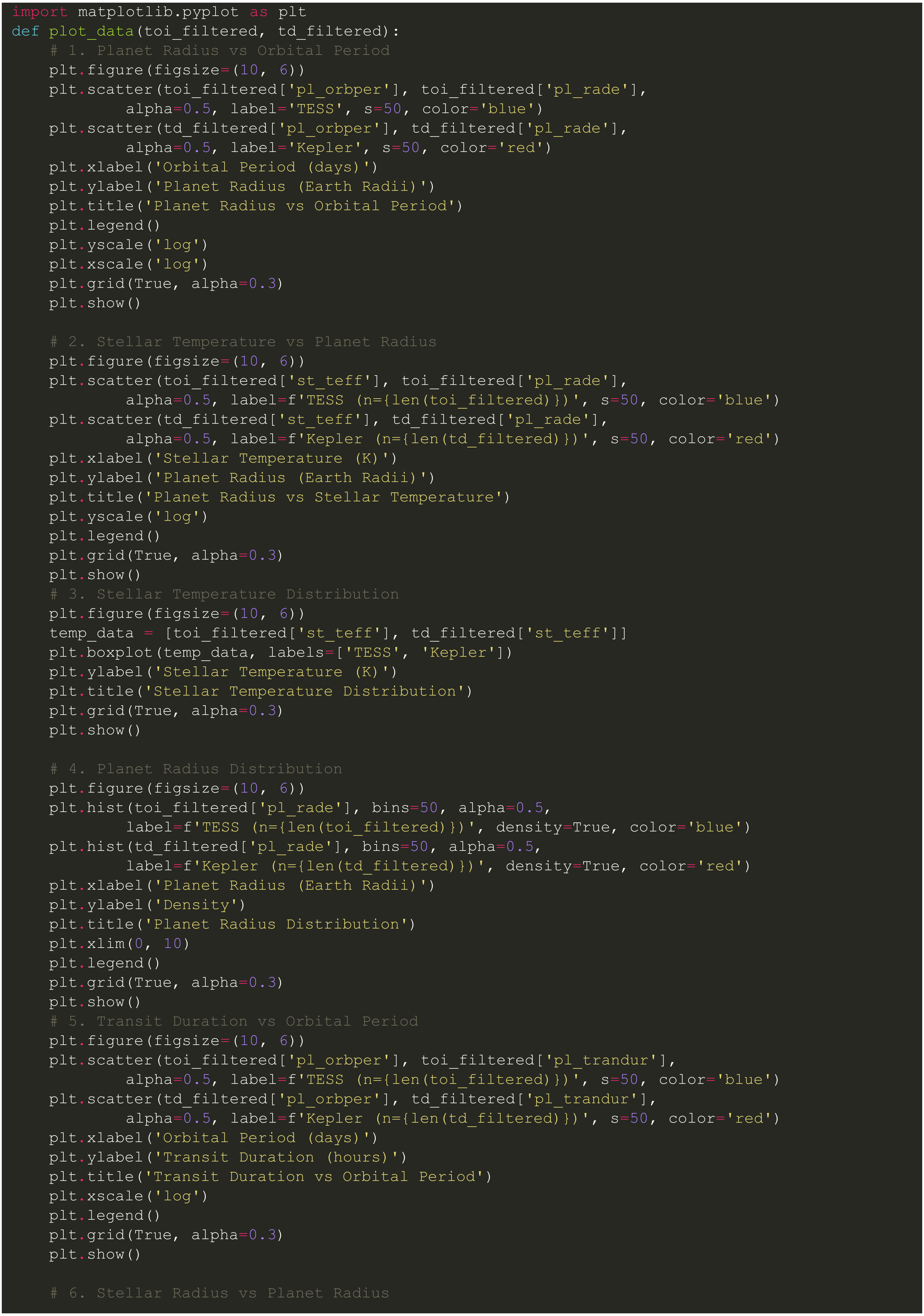


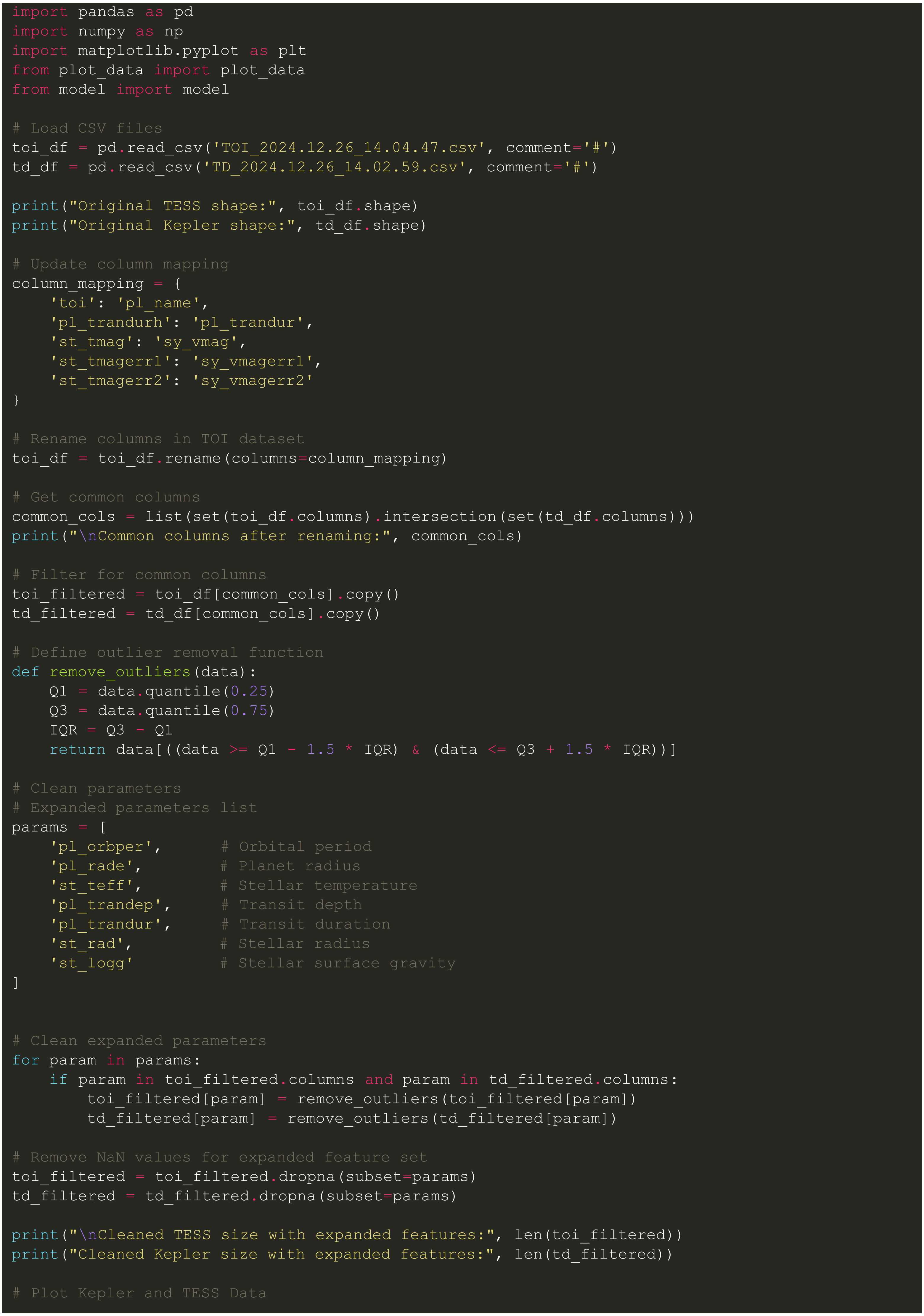
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report, confusion_matrix, roc_curve, auc
from sklearn.model_selection import train_test_split
import seaborn as sns
import matplotlib.pyplot as plt


def model(toi_filtered, td_filtered):
    # Updated features list to match available columns
    features = [
        'pl_orbper',      # Orbital period
        'pl_rade',        # Planet radius
        'st_teff',        # Stellar temperature
        'pl_trandep',     # Transit depth
        'pl_trandur',     # Transit duration
        'st_rad',         # Stellar radius
        'st_logg'         # Stellar surface gravity
    ]


    # Prepare positive class (confirmed Kepler planets)
    X_positive = td_filtered[features].values
    y_positive = np.ones(len(X_positive))

    # Create synthetic negative class by perturbing the positive examples
    np.random.seed(42)
    X_negative = X_positive * np.random.uniform(0.5, 1.5, size=X_positive.shape)
    y_negative = np.zeros(len(X_negative))

    # Combine positive and negative examples
    X_train = np.vstack([X_positive, X_negative])
    y_train = np.hstack([y_positive, y_negative])

    # Prepare TESS testing data
    X_test = toi_filtered[features].values

    # Scale features
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)
    # Train Random Forest model
    rf_model = RandomForestClassifier(
        n_estimators=1000,
        max_depth=15,
        min_samples_split=5,
        min_samples_leaf=2,
        random_state=42,
        n_jobs=-1
    )



    # Cross-validation scores
    cv_scores = cross_val_score(rf_model, X_train_scaled, y_train, cv=5)
    print("\nCross-validation scores:", cv_scores)
    print(f"Mean CV score: {cv_scores.mean():.4f}")
    print(f"Standard deviation: {cv_scores.std():.4f}")

    # Train final model
    # Split data into train and validation sets
    X_train_split, X_val, y_train_split, y_val = train_test_split(
        X_train_scaled, y_train, test_size=0.2, random_state=42
)

# Train model on training set
rf_model.fit(X_train_split, y_train_split)

# Calculate accuracies
train_accuracy = rf_model.score(X_train_split, y_train_split)
val_accuracy = rf_model.score(X_val, y_val)

print(f"\nTraining Accuracy: {train_accuracy:.4f}")
print(f"Validation Accuracy: {val_accuracy:.4f}")

# Get predictions
train_preds = rf_model.predict(X_train_scaled)
train_probs = rf_model.predict_proba(X_train_scaled)[:, 1]

# Print classification report
print("\nClassification Report:")
print(classification_report(y_train, train_preds))

# Confusion Matrix
cm = confusion_matrix(y_train, train_preds)
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, cmap='Blues', fmt='d')
plt.title('Confusion Matrix')

plt.xticks([0.5, 1.5], ['Not Planet', 'Planet'], rotation=45, ha='center')

plt.yticks([0.5, 1.5], ['Not Planet', 'Planet' ], rotation=45, va='center')
# Bold each x and y label
# Show xlabel

plt.xlabel('Predicted Label', fontweight='bold')
plt.ylabel('True Label', fontweight='bold')


plt.show()
# ROC Curve
fpr, tpr, _ = roc_curve(y_train, train_probs)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(10, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2,
        label=f'ROC curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()

# Feature importance
importance = pd.DataFrame({
    'Feature': features,
    'Importance': rf_model.feature_importances_
})
importance = importance.sort_values('Importance', ascending=False)

plt.figure(figsize=(10, 6))
plt.bar(importance['Feature'], importance['Importance'])
plt.title('Feature Importance')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

```
   # Get TESS predictions (probabilities)
    tess_probs = rf_model.predict_proba(X_test_scaled)[:, 1]  # Get probability of class 1
    toi_filtered['rf_probability'] = tess_probs
    print("\nPrediction statistics:")
    print(f"Shape: {tess_probs.shape}")
    print(f"Min: {tess_probs.min()}, Max: {tess_probs.max()}")
    print(f"Mean: {tess_probs.mean()}, Std: {tess_probs.std()}")
    # Show top candidates
    print("\nTop TESS Candidates (Random Forest):")
    print(toi_filtered.sort_values('rf_probability', ascending=False)[['pl_name',
'rf_probability']].head(20))
```



```python
import matplotlib.pyplot as plt
def plot_data(toi_filtered, td_filtered):
    # 1. Planet Radius vs Orbital Period
    plt.figure(figsize=(10, 6))
    plt.scatter(toi_filtered['pl_orbper'], toi_filtered['pl_rade'],
            alpha=0.5, label='TESS', s=50, color='blue')
    plt.scatter(td_filtered['pl_orbper'], td_filtered['pl_rade'],
            alpha=0.5, label='Kepler', s=50, color='red')
    plt.xlabel('Orbital Period (days)')
    plt.ylabel('Planet Radius (Earth Radii)')
    plt.title('Planet Radius vs Orbital Period')
    plt.legend()
    plt.yscale('log')
    plt.xscale('log')
    plt.grid(True, alpha=0.3)
    plt.show()

    # 2. Stellar Temperature vs Planet Radius
    plt.figure(figsize=(10, 6))
    plt.scatter(toi_filtered['st_teff'], toi_filtered['pl_rade'],
            alpha=0.5, label=f'TESS (n={len(toi_filtered)})', s=50, color='blue')
    plt.scatter(td_filtered['st_teff'], td_filtered['pl_rade'],
            alpha=0.5, label=f'Kepler (n={len(td_filtered)})', s=50, color='red')
    plt.xlabel('Stellar Temperature (K)')
    plt.ylabel('Planet Radius (Earth Radii)')
    plt.title('Planet Radius vs Stellar Temperature')
    plt.yscale('log')
    plt.legend()
    plt.grid(True, alpha=0.3)
    plt.show()
    # 3. Stellar Temperature Distribution
    plt.figure(figsize=(10, 6))
    temp_data = [toi_filtered['st_teff'], td_filtered['st_teff']]
    plt.boxplot(temp_data, labels=['TESS', 'Kepler'])
    plt.ylabel('Stellar Temperature (K)')
    plt.title('Stellar Temperature Distribution')
    plt.grid(True, alpha=0.3)
    plt.show()

    # 4. Planet Radius Distribution
    plt.figure(figsize=(10, 6))
    plt.hist(toi_filtered['pl_rade'], bins=50, alpha=0.5,
            label=f'TESS (n={len(toi_filtered)})', density=True, color='blue')
    plt.hist(td_filtered['pl_rade'], bins=50, alpha=0.5,
            label=f'Kepler (n={len(td_filtered)})', density=True, color='red')
    plt.xlabel('Planet Radius (Earth Radii)')
    plt.ylabel('Density')
    plt.title('Planet Radius Distribution')
    plt.xlim(0, 10)
    plt.legend()
    plt.grid(True, alpha=0.3)
    plt.show()
    # 5. Transit Duration vs Orbital Period
    plt.figure(figsize=(10, 6))
    plt.scatter(toi_filtered['pl_orbper'], toi_filtered['pl_trandur'],
            alpha=0.5, label=f'TESS (n={len(toi_filtered)})', s=50, color='blue')
    plt.scatter(td_filtered['pl_orbper'], td_filtered['pl_trandur'],
            alpha=0.5, label=f'Kepler (n={len(td_filtered)})', s=50, color='red')
    plt.xlabel('Orbital Period (days)')
    plt.ylabel('Transit Duration (hours)')
    plt.title('Transit Duration vs Orbital Period')
    plt.xscale('log')
    plt.legend()
    plt.grid(True, alpha=0.3)
    plt.show()

    # 6. Stellar Radius vs Planet Radius
```

```
plt.figure(figsize=(10, 6))
plt.scatter(toi_filtered['st_rad'], toi_filtered['pl_rade'],
        alpha=0.5, label='TESS', s=50, color='blue')
plt.scatter(td_filtered['st_rad'], td_filtered['pl_rade'],
        alpha=0.5, label='Kepler', s=50, color='red')
plt.xlabel('Stellar Radius (Solar Radii)')
plt.ylabel('Planet Radius (Earth Radii)')
plt.title('Planet Size vs Star Size')
plt.yscale('log')
plt.xscale('log')
plt.legend()
plt.grid(True, alpha=0.3)
plt.show()

# 7. Surface Gravity vs Transit Depth
plt.figure(figsize=(10, 6))
plt.scatter(toi_filtered['st_logg'], toi_filtered['pl_trandep'],
        alpha=0.5, label='TESS', s=50, color='blue')
plt.scatter(td_filtered['st_logg'], td_filtered['pl_trandep'],
        alpha=0.5, label='Kepler', s=50, color='red')
plt.xlabel('Stellar Surface Gravity (log g)')
plt.ylabel('Transit Depth')
plt.title('Transit Depth vs Surface Gravity')
plt.yscale('log')
plt.legend()
plt.grid(True, alpha=0.3)
plt.show()

# 8. Feature Correlation Matrix
features = ['pl_orbper', 'pl_rade', 'st_teff', 'pl_trandep',
            'pl_trandur', 'st_rad', 'st_logg']

# TESS correlations
plt.figure(figsize=(12, 10))
correlation_tess = toi_filtered[features].corr()
plt.imshow(correlation_tess, cmap='coolwarm', aspect='auto')
plt.colorbar()
plt.xticks(range(len(features)), features, rotation=45)
plt.yticks(range(len(features)), features)
plt.title('TESS Feature Correlations')
for i in range(len(features)):
    for j in range(len(features)):
        plt.text(j, i, f'{correlation_tess.iloc[i, j]:.2f}',
                ha='center', va='center')
plt.tight_layout()
plt.show()
```



```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from plot_data import plot_data
from model import model

# Load CSV files
toi_df = pd.read_csv('TOI_2024.12.26_14.04.47.csv', comment='#')
td_df = pd.read_csv('TD_2024.12.26_14.02.59.csv', comment='#')

print("Original TESS shape:", toi_df.shape)
print("Original Kepler shape:", td_df.shape)

# Update column mapping
column_mapping = {
    'toi': 'pl_name',
    'pl_trandurh': 'pl_trandur',
    'st_tmag': 'sy_vmag',
    'st_tmagerr1': 'sy_vmagerr1',
    'st_tmagerr2': 'sy_vmagerr2'
}

# Rename columns in TOI dataset
toi_df = toi_df.rename(columns=column_mapping)

# Get common columns
common_cols = list(set(toi_df.columns).intersection(set(td_df.columns)))
print("\nCommon columns after renaming:", common_cols)

# Filter for common columns
toi_filtered = toi_df[common_cols].copy()
td_filtered = td_df[common_cols].copy()

# Define outlier removal function
def remove_outliers(data):
    Q1 = data.quantile(0.25)
    Q3 = data.quantile(0.75)
    IQR = Q3 - Q1
    return data[((data >= Q1 - 1.5 * IQR) & (data <= Q3 + 1.5 * IQR))]

# Clean parameters
# Expanded parameters list
params = [
    'pl_orbper',       # Orbital period
    'pl_rade',         # Planet radius
    'st_teff',         # Stellar temperature
    'pl_trandep',      # Transit depth
    'pl_trandur',      # Transit duration
    'st_rad',          # Stellar radius
    'st_logg'          # Stellar surface gravity
]


# Clean expanded parameters
for param in params:
    if param in toi_filtered.columns and param in td_filtered.columns:
        toi_filtered[param] = remove_outliers(toi_filtered[param])
        td_filtered[param] = remove_outliers(td_filtered[param])

# Remove NaN values for expanded feature set
toi_filtered = toi_filtered.dropna(subset=params)
td_filtered = td_filtered.dropna(subset=params)

print("\nCleaned TESS size with expanded features:", len(toi_filtered))
print("Cleaned Kepler size with expanded features:", len(td_filtered))

# Plot Kepler and TESS Data
```

```
plot_data(toi_filtered, td_filtered)

model(toi_filtered, td_filtered)
```