A3 report

Q1

Task (a): Monthly Returns and Summary Statistics

1. **Import Libraries and Load Data**
   - Libraries: `pandas` for data manipulation, `numpy` for numerical operations, and `matplotlib.pyplot` for plotting.
   - The Excel file is loaded into a DataFrame (`df`) using `pd.read_excel()` with the file path specified.
2. **Filter CSI 300 Data**
   - The dataset contains multiple indices (e.g., "000001", "000300"). We filter for the CSI 300 index by selecting rows where `Indexcd == '000300'`, creating `csi300_df`.
   - This ensures we analyze only the relevant index.
3. **Convert Trading Date to Datetime**
   - The `Trddt` column is converted to a `datetime` format using `pd.to_datetime()` to enable time-based operations.
   - A warning about `SettingWithCopyWarning` appears due to modifying a slice of the DataFrame. This could be avoided with `.loc`, but it doesn't affect the result here.
4. **Set Trading Date as Index**
   - The `Trddt` column is set as the DataFrame index using `set_index()`, facilitating time-series operations like resampling.
5. **Resample to Monthly Closing Indices**
   - The daily closing indices (`Clsindex`) are resampled to monthly frequency using `.resample('M').last()`, taking the last trading day's closing value of each month.
   - Note: `'M'` is deprecated in favor of `'ME'` (month-end), but it works here. This creates `monthly_closes`.
   - The first few values (e.g., 1009.597 for January 2006) confirm the resampling.
6. **Compute Monthly Returns**
   - Monthly returns are calculated using the formula $R_{k,t} = \frac{I_{k,t}}{I_{k,t-1}} - 1$, implemented via `monthly_closes.pct_change()`.
   - The first return (January 2006) is `NaN` because there's no prior month, as expected.
7. **Remove Missing Values**
   - The `NaN` value is dropped using `.dropna()`, leaving 215 monthly returns (February 2006 to December 2023, assuming complete data).
8. **Calculate Summary Statistics**
   - **Mean**: Average monthly return, computed with `.mean()`.
   - **Standard Deviation**: Measure of return volatility, computed with `.std()`.
   - **Skewness**: Asymmetry of the return distribution, computed with `.skew()`.
   - **Kurtosis**: Tailedness of the distribution, computed with `.kurt()` (excess kurtosis relative to a normal distribution's kurtosis of 3).
   - Results are printed and organized in a table using a `pandas` DataFrame.

Results

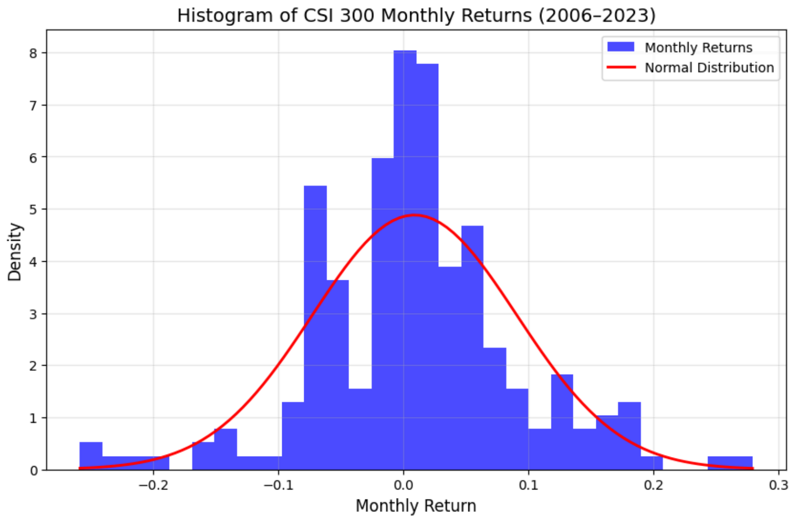
- **Mean Monthly Return**: 0.009042 (0.9042% per month)
  - Suggests a positive average monthly growth over the period.
- **Standard Deviation**: 0.081745 (8.1745% per month)
  - Indicates significant volatility in monthly returns.
- **Skewness**: 0.018437 (near zero, slightly positive)
  - Iimplies the distribution is nearly symmetric, with a slight right tilt.
- **Kurtosis**: 1.455060 (positive excess kurtosis, leptokurtic)
  - Suggests a leptokurtic distribution with heavier tails than a normal distribution, indicating more frequent extreme returns.

Task (b): Histogram of Monthly Returns

1. **Set Up Plot**
   - A figure size of 10x6 inches is set with `plt.figure(figsize=(10, 6))` for clarity.
2. **Plot Histogram**
   - The histogram is created using `plt.hist()` with:
     - `bins=30`: Provides granularity for ~215 data points (roughly 7-8 points per bin).
     - `density=True`: Normalizes the histogram to a probability density, allowing comparison with a normal curve.
     - `alpha=0.7`: Transparency for visual appeal.
     - `color='blue'`: Distinguishes the histogram bars.
3. **Overlay Normal Distribution**
   - A normal distribution curve is overlaid using `scipy.stats.norm.pdf()`, parameterized by the sample mean (`mu`) and standard deviation (`sigma`).

- This curve (red line) serves as a visual benchmark for normality.

4. **Add Labels and Formatting**
   - Title, x-label ("Monthly Return"), y-label, and legend are added for context.
   - A light grid (`alpha=0.3`) enhances readability.



Task (c): Normality Assessment with Shapiro-Wilk Test

1. **Import Required Library**
   - `scipy.stats` is imported for the `shapiro` function.
2. **Perform Shapiro-Wilk Test**
   - The test is applied to `monthly_returns` using `shapiro()`, returning:
     - **Test Statistic (W)**: Measures how closely the sample matches a normal distribution (0 to 1, closer to 1 is more normal).
     - **P-value**: Probability of observing the statistic under the null hypothesis (data is normal).
3. **Interpret Results**
   - Detailed explanation of the test is provided in markdown:
     - $W = 0.971284$: Close to 1 but not perfect.
     - $P - value = 0.000228$: Far below 0.05.

Results

- **Shapiro-Wilk Test Statistic**: 0.971284
- **P-value**: 0.000228

Interpretation

- **Test Statistic**: A value of 0.971284 is relatively high, suggesting the distribution isn't drastically non-normal, but it's not 1 (perfect normality).
- **P-value**: At 0.000228 (< 0.05), we reject the null hypothesis at the 5% significance level. This indicates strong evidence that the CSI 300 monthly returns are not normally distributed.
- **Sample Size Consideration**: With ~215 observations, the test is sensitive to small deviations from normality, amplifying the significance of the low p-value.

Conclusion

The Shapiro-Wilk test, combined with the histogram and kurtosis, suggests the CSI 300 monthly returns deviate from normality. The positive kurtosis (heavier tails) and slight skewness likely contribute to this rejection, indicating a distribution with more extreme values than a normal one.

Q2

1. **Import Libraries**
   - `pandas` for data manipulation, `numpy` for calculations, `statsmodels.api` for regressions, `matplotlib.pyplot` for plotting, and `tabulate` for table formatting.
2. **Load Data**
   - Two Excel files (`TRD_Week_1.xlsx` and `TRD_Week_2.xlsx`) are read using `pd.read_excel()`, skipping metadata rows and ensuring `Stkcd` is a string.
   - Files are concatenated into `stock_returns` using `pd.concat()`.
3. **Filter Time Period**

- Data is filtered to include only weeks before "2023-01" using a condition on `Trdwnt`, resetting the index for consistency.

4. **Output**
   - A sample of the merged data (with risk-free rates, added later) is displayed using `tabulate` for verification.

Task (b): Calculate Weekly Market Returns

1. **Group and Average**
   - `data.groupby('Trdwnt')['Wretnd'].mean()` calculates the mean return across all stocks for each week, stored in `market_returns`.
   - The column is renamed to `Market_Return`.
2. **Merge with Main Dataset**
   - Market returns are merged back into the main dataset (`data`) using `pd.merge()` on `Trdwnt`.
3. **Output**
   - A sample of the updated dataset with `Market_Return` is printed (e.g., 0.016611 for "2017-01").

Results

Market returns represent a proxy for the market portfolio's performance, consistent with CAPM's $r_{m,t}$.

```
Data with Market Returns:
+---+--------+---------+-----------+-----------------------+------------------------+
|   | Stkcd  | Trdwnt  |  Wretnd   |   risk_free_return    |     Market_Return      |
+---+--------+---------+-----------+-----------------------+------------------------+
| 0 | 000001 | 2017-01 | 0.003297  | 0.0005714053404517472 |  0.016611333917923535  |
| 1 | 000001 | 2017-02 | 0.003286  | 0.0007376706344075501 |  -0.035047628272251 31 |
| 2 | 000001 | 2017-03 |  0.00655  | 0.0005432766629382968 | -0.017416864300626302  |
| 3 | 000001 | 2017-04 | 0.011931  | 0.0007736414554528892 |  0.019901674732111994  |
| 4 | 000001 | 2017-05 | -0.007503 | 0.0008020293316803873 | -0.0025298505029483177 |
+---+--------+---------+-----------+-----------------------+------------------------+
```

Task (c): Load Weekly Risk-Free Return Data

1. **Load Risk-Free Data**
   - `weekly_risk_free_rate.xlsx` is read into `rf_data` using `pd.read_excel()`.
2. **Format Trading Week**
   - `trading_date_yw` is converted to `datetime` using `pd.to_datetime()`, set as the index, and reformatted to a year-week string (e.g., "2017-01") using `.to_period('W').strftime('%Y-%U')`.
3. **Merge with Stock Data**
   - Risk-free rates are merged into `data` using `pd.merge()` on `Trdwnt`.
4. **Output**
   - The merged sample shows risk-free rates alongside stock returns (e.g., 0.000571 for "2017-01").

Results

The risk-free rate ($r_{f,t}$) is successfully integrated, enabling excess return calculations.

Task (d): Replicate Tables 2 and 3 from Chen et al. (2019)

Step 1: Ensure Data Consistency

- **Filter Consistent Stocks**: Only stocks present in all weeks are retained using `set.intersection()` on grouped `Stkcd` sets, resulting in 1604 stocks.
- **Output**: Number of consistent stocks is printed (1604).

Step 2: Divide Data into Three Periods

- **Split Weeks**: 308 unique weeks are divided into three periods (102, 102, 104 weeks) using integer division and slicing of `unique_weeks`.
- **Assign Data**: Each period's data (`period1_data`, `period2_data`, `period3_data`) is created by filtering `data` based on `Trdwnt`.
- **Output**: Week counts per period are verified (102, 102, 104).

Step 3: Calculate Individual Stock Betas (Period 1)

- **Model**: $r_{i,t} = \alpha_i + \beta_i r_{m,t} + \epsilon_i$.
- **Function**: `calculate_stock_beta()` uses `sm.OLS` to regress stock returns (`Wretnd`) on market returns (`Market_Return`) with a constant, returning β_i.
- **Execution**: Applied to `period1_data` via `groupby('Stkcd')`, storing results in `stock_betas`.

Step 4: Form Ten Portfolios

- **Sort and Group**: Stocks are sorted into deciles based on β_i using `pd.qcut()` (10 groups, labeled 1–10).
- **Merge**: Beta groups are merged into `period2_data` and `period3_data`.
- **Output**: Stock counts per portfolio are printed (e.g., 161, 160, etc.), confirming near-equal distribution.

```
Number of Stocks in Each Portfolio:
+---+-----------+-------------+
|   | Portfolio | Stock Count |
+---+-----------+-------------+
| 0 |     1     |     161     |
| 1 |     2     |     160     |
| 2 |     3     |     160     |
| 3 |     4     |     161     |
| 4 |     5     |     160     |
| 5 |     6     |     160     |
| 6 |     7     |     161     |
| 7 |     8     |     160     |
| 8 |     9     |     160     |
| 9 |    10     |     161     |
+---+-----------+-------------+
```

Step 5: Calculate Portfolio Betas (Period 2) - Table 2

- **Portfolio Returns**: Equal-weighted returns are computed per portfolio and week using `groupby(['Trdwnt', 'beta_group'])['Wretnd'].mean()`.
- **Excess Returns**: Portfolio excess returns ($r_{p,t} - r_{f,t}$) and market excess returns ($r_{m,t} - r_{f,t}$) are calculated.
- **Model**: $r_{p,t} - r_{f,t} = \alpha_p + \beta_p(r_{m,t} - r_{f,t}) + \epsilon_{p,t}$.
- **Regression**: For each portfolio (1–10), `sm.OLS` regresses excess portfolio returns on excess market returns, storing α_p, β_p, t-values, p-values, and R^2.
- **Output**: Results are formatted into `table2_df` and printed, matching Table 2's structure:
  - β_p: 0.673459 to 1.17407, increasing across portfolios.
  - α_p: Mostly insignificant (p > 0.05, except Portfolio 9 at 0.0901).
  - R^2: High (0.8067–0.9802), indicating strong explanatory power.

```
Table 2: Time Series Regression Results for Portfolios
+-----------+----------+---------+---------+----------+--------+------------+-----------+
| Portfolio |  Alpha   | Alpha_t | Alpha_p |   Beta   | Beta_t |   Beta_p   | R_squared |
+-----------+----------+---------+---------+----------+--------+------------+-----------+
|     1     | -3.6e-05 | -0.033  | 0.9738  | 0.673459 | 20.328 | 4.1086e-37 |  0.8067   |
|     2     | 0.000628 |  0.786  | 0.4335  | 0.806966 | 32.951 | 3.6391e-55 |  0.9164   |
|     3     | 0.000533 |  0.743  | 0.4591  | 0.841312 | 38.289 | 3.7288e-61 |  0.9367   |
|     4     | 0.000559 |  0.789  | 0.4318  | 0.892307 | 41.089 | 5.1897e-64 |  0.9446   |
|     5     | 6.8e-05  |  0.116  | 0.9081  | 0.934127 | 52.07  | 9.5865e-74 |  0.9648   |
|     6     | 0.000441 |  0.841  | 0.4024  | 1.011989 | 62.973 | 1.1111e-81 |  0.9756   |
|     7     | 0.000372 |  0.796  | 0.4277  | 1.002425 | 69.98  | 4.0619e-86 |  0.9802   |
|     8     | 0.000385 |  0.722  | 0.4719  | 1.067248 | 65.332 | 3.1736e-83 |  0.9773   |
|     9     | 0.001052 |  1.712  | 0.0901  | 1.129976 | 59.964 | 1.2530e-79 |  0.9732   |
|    10     | 0.000511 |  0.618  | 0.5380  | 1.17407  | 46.366 | 5.9601e-69 |   0.956   |
+-----------+----------+---------+---------+----------+--------+------------+-----------+
```

Step 6: Calculate Average Excess Returns (Period 3)

- **Portfolio Returns**: Equal-weighted returns are computed for Period 3.
- **Excess Returns**: $r_{p,t} - r_{f,t}$ is calculated per week.
- **Average**: Mean excess returns per portfolio are computed using `groupby('beta_group')`.
- **Output**: Ranges from 0.000423 (Portfolio 3) to 0.001716 (Portfolio 5).

```
Average Excess Returns for Each Portfolio in Period 3:
+---+-----------+----------------------+
|   | Portfolio |  Avg Excess Return   |
+---+-----------+----------------------+
```

```
| 0 |    1.0     | 0.0011227736376281361 |
| 1 |    2.0     | 0.0009735728673803038 |
| 2 |    3.0     | 0.0004225399465882249 |
| 3 |    4.0     | 0.001097632133415603  |
| 4 |    5.0     | 0.001715878003518918  |
| 5 |    6.0     | 0.0010043155035189177 |
| 6 |    7.0     | 0.0014534650465205781 |
| 7 |    8.0     | 0.001166215255994165  |
| 8 |    9.0     | 0.001558971196588225  |
| 9 |   10.0     | 0.0011130892393746462 |
+---+-----------+-----------------------+
```

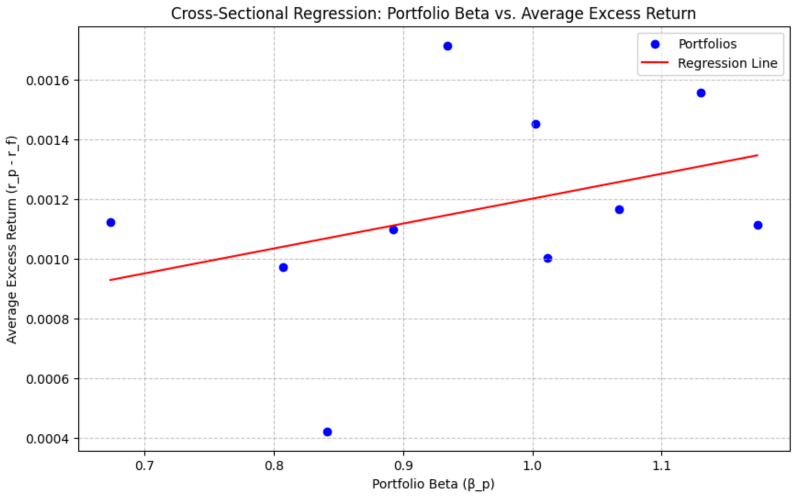
Step 7: Cross-Sectional Regression - Table 3

- **Model**: $\overline{r_{p,t} - r_{f,t}} = \gamma_0 + \gamma_1 \beta_p + \epsilon_p$.
- **Data**: Merges Period 3 average excess returns with Period 2 β_p.
- **Regression**: `sm.OLS` regresses average excess returns on β_p.
- **Output**: Results are formatted into Table 3:
  - γ_0: 0.000367 (t = 0.496, p = 0.6334, insignificant).
  - γ_1: 0.000835 (t = 1.087, p = 0.3088, insignificant).
  - R^2: 0.1287.
  - F-statistic: 1.1813 (p = 0.3088).

```
Table 3: Cross-Sectional Regression Results
+-------------+----------+---------+---------+-----------+-------------+-----------+
| Coefficient | Estimate | t-value | p-value | R-squared | F-statistic | F p-value |
+-------------+----------+---------+---------+-----------+-------------+-----------+
|   gamma_0   | 0.000367 |  0.496  | 0.6334  |  0.1287   |   1.1813    |  0.3088   |
|   gamma_1   | 0.000835 |  1.087  | 0.3088  |           |             |           |
+-------------+----------+---------+---------+-----------+-------------+-----------+
```

Step 8: Visualization

- **Scatter Plot**: Plots β_p vs. average excess returns with a regression line, showing a weak positive trend.



Results Interpretation

- **Table 2**: Portfolio β_p increases from 0.673 (Portfolio 1) to 1.174 (Portfolio 10), reflecting higher systematic risk. Most α_p values are insignificant (p > 0.05), aligning with CAPM's prediction of zero alpha, though high R^2 (0.8067–0.9802) suggests market returns explain most variance. Unlike Chen et al. (2019), fewer α_p are significant, possibly due to a larger sample (1604 vs. 50 stocks).
- **Table 3**: The insignificant γ_1 (p = 0.3088) indicates no strong linear relationship between β_p and excess returns, contradicting CAPM's expectation of a positive risk-return tradeoff. The low R^2 (0.1287) suggests other factors influence returns, consistent with Chen et al.'s findings of limited CAPM applicability in China.