File Name	Local Code	Correct 2 Tokens fwd	CodeGPT Adapted Prediction	CodeGPT Prediction	Kite Prediction	Tabnine Prediction
Django1_cut1.py	return Question.objects.filter(pub_date__lte=timezone.now()).	order_by('-pub_date'	.order_by('-pub_date'	.order_by	order_by('-created_at')	order_by(' published
django1_cut2.py	def vote(request, question_id): question = get_	get_object_or_404(Question	get_object_or_404(Question	get_object_or_404(Question	get_object_or_404(Question	get_queryset(, N/A
django1_cut3.py	def vote(request, question_id): question = get_object_or_404(Question, pk=question_id) try: selected_choice = question.	choice_set.get(	choice_set.filter	.choice except	choice(question_id)	now , N/A
plotly1_cut1.py	# Load data df = pd.read_csv('data/wellspublic.csv') df['Date_Well_Completed'] = pd.	to_datetime(, df['Date_well_completed']	to_datetime(, df['Date_Well_Completed'	.to_datetime	.core	read_, df['Date_Well_Completed']
plotly1_cut2.py	trim = df[['API_WellNo', 'Well_Type', 'Well_Name']] trim.index = trim['API_WellNo'] dataset = trim.	to_dict(, orient = 'index')	drop_duplicates, return{'API_Wells'}	.index.astype	.index	Dash, __name__
plotly1_cut3.py	@app.callback(Output('well_statuses', 'value'), [Input('well_status_selector', 'value')]) def display_status(selector): if selector == 'all': return list(	WELL_STATUSES.keys()	Output('Wells"}	filter(lambda	...	df['Date_Well_Completed'].values()
pandas1_cut1.py	tmp = tmp.applymap(lambda x: 1 if x > prof_thresh else 0) rev = pd.DataFrame({'idx': df.index, 'curr' : df[target].values}) rev.set_index('idx', drop = True, inplace = True) window_vec = [fwd_bars, fwd_bars*2,int(fwd_bars/2)] name_vec = ['sma1','sma2','sma3'] rev = fg.add_many_smas(rev, 'curr', window_vec, name_vec) rev = rev.apply(above_curr, axis=1, result_type='expand') df['rev'] = np.multiply(rev.sum(axis=1) >= 2, 1) df['bogsum'] = tmp.	sum(axis =1)	sum(axis=1).item()	.iloc[:,	extend(iterable)	sum(axis=1)
pandas1_cut2.py	tmp.set_index('idx', drop=True, inplace=True) # calculating percent change for i in range(fwd_bars): tmp[i] = tmp.curr.	percent_change(i)	data[i, 'idx']	.get('	sum(axis=1)	sum(i)
pandas1_cut3.py	def above_curr(row, comp_col='curr'): tmp = [] for i in range(1, len(row)): tmp.append(row[i] >	row[comp_col]	tmp[-1] return tmp	0)	0	comp_col
flask2_cut1.py	@jwt_optional @marshal_with(profile_schema) get_profile(username): user = User.query.	filter_by(username = username)	.get(username)	filter_by(username=username	filter_by(username	get(username
flask2_cut2.py	def follow_user(username): user = User.query.filter_by(username=username).first() if not user: raise InvalidUsage.user_not_found() current_user.profile.	.follow(user.profile)	.user=user	add_user(user)	follow(username	.follow = user
flask2_cut3.py	def unfollow_user(username): user = User.query.filter_by(username=username).first() if not user: raise InvalidUsage.user_not_found() current_user.profile.	unfollow(user.profile)	.profile_name = '{	unfollow(username) return	save()	unfollow_user(username)
sklearn1_cut1.py	def ingest_augmentor(self, augmentor, raw_df): self.augmentor = augmentor self.augmentor.augmented_df = self.augmentor.	make_augmented_df(raw_df)	.augment(raw_df	.apply	augmented_df	augmented_df
sklearn1_cut2.py	self.split_data[key].loc[:, self.feat_list] self.xy_data[key]['_y'] = self.split_data[key].loc[:, self.	.bogey]	.feat_list	.split	feat_list	.feat_list
sklearn1_cut3.py	def predict(self, df): feat_df = self.get_feats(df) scaled_df = self.scaler.scale(feat_df) return self.obj.	.predict(scaled_df)	.predict(scaled_df)	.fit(	.predict(scaled_df)	.predict(scaled_df)
tensorflow1_cut1.py	def logit(z): return 1 / (1 + np.	.exp(-z)	.exp(-z , [**2) * tf.log(1+np.exp(-z**2)]	.exp(-z))	pi*z	.log(z)
tensorflow1_cut2.py	y = tf.placeholder(tf.int32, shape=(None), name='y') with tf.name_scope('dnn'): hidden1 = tf.layers.	.dense(X, n_hidden1	.dense(y	.dense(x	.dense(X	.hidden
tensorflow1_cut3.py	logits = tf.layers.dense(hidden2, n_outputs, name='outputs') xentropy = tf.nn.	sparse_softmax_cross_entropy_with_logits	softmax_cross_entropy_with_logits(labels, logits	sparse_softmax_	softmax(logits)	.softmax(xentropy)

pandas2_cut1.py	def add_ema_custom (df, colname = 'close_price', window = 6, feature_name = '1min'): df [feature_name] = df.loc[:,colname].	ewm(span=window	.groupby(level	.apply(lambda x:	.loc[:,colname	value
pandas2_cut2.py	def add_sma_custom (df, colname = 'close_price', window = 6, feature_name = '1min'): df [feature_name] = df.loc[:,colname].	rolling(window=window	.sum(axis	.apply(	.ewm(span=window ***this shows u	value
pandas2_cut3.py	tmp_p[str(i)] = df.pos_bar.rolling (window=i, min_periods=1).sum() * df.pos_bar.shift(i) tmp_p['streak'] = tmp_p.	.apply(check_across	N/A	.get('streak', 0)	.streak(window	.sum()*df.
plotly2_cut1.py	contours = [] def pairwise(iterable): a = iter(iterable) return izip(a, a) i = 0 for lat, lon in pairwise(df. columns): contours.	.append(dict(	.append([lat	.append(df[	.append(object	append(lat
plotly2_cut2.py	import plotly_express as px #making scatter fig = px.	scatter(	.figure(figsize=(3,3))	.Figure(figsize = (8,	fig	.figure(fig
plotly2_cut3.py	# plot this data df.iplot(kind=	scatter', mode = 'markers'	kind, title = title, xlabel = "Places"	kind, color=color, marker=marker	1	data', name =
sklearn2_cut1.py	from sklearn.base import BaseEstimator, TransformerMixin from sklearn.linear_model import LinearRegression class PrecipitationTransformer (BaseEstimator, TransformerMixin): def fit(self,	X, y):	X):	X, y	X,y	data):
sklearn2_cut2.py	y = antelope_df ['spring_fawn_count'] # Step 2: train-test split X_train, X_test, y_train, y_test =	train_test_split(X,y	xtest_split n_train = tf.shape(data)[0] def loss(y, x): re	arima_	train_test_split(X	split(
sklearn2_cut3.py	# Step 3: fit preprocessor pipeline pipe = Pipeline(	steps = [	it.imap(partial(step_work, preprocessor=self.pipeline	self.pipeline_config) pipeline.fit	X,y	X
tensorflow2_cut1.py	frames.append(img) if step % n_change_steps == 0: action = env. action_space.sample() # play randomly obs, reward, done, info = env.	step(action)	step(action) if done or info['re ...	step(obs, action, done, info)	step(0)	step(step)
tensorflow2_cut2.py	X = tf.placeholder(tf.float32, shape= [None, n_inputs]) hidden = tf.layers. dense(X, n_hidden, activation=tf.	tf.nn.elu	tf.nn.relu)	tf.nn.relu	tf.nn.softmax	tf.constant()
tensorflow2_cut3.py	grads_and_vars_feed.append ((gradient_placeholder, variable)) training_op = optimizer.	apply_gradients(grads_and_vars_feed)	apply_gradients(data, update_ops = training_op	minimize(loss, global_	minimize(loss)	compute_gradients(training_